Ecole Nationale Polytechnique

Département d'électronique

Thèse de Doctorat d'Etat
en Electronique

Présentée par: Ahmed Riadh BABA-ALI
Magister en Cybernetique

Thème

Optimisation des performances temporelles de circuits intégrés VLSI

Thèse soutenue publiquement le 18/03/1998 devant le jury:

Président :	**A. Kheladi**	**Professeur USTHB**
Rapporteur :	**A. Farah**	**Maître de conférences ENP**
Examinateur :	**M. Haddadi**	**Maître de conférences ENP**
Examinateur :	**M. Selmane**	**Professeur USTHB**
Examinateur :	**M. Mehenni**	**Maître de conférences ENP**

Ecole Nationale Polytechnique

Département d'électronique

Thèse de Doctorat d'Etat en Electronique

Présentée par: Ahmed Riadh BABA-ALI
Magister en Cybernetique

Thème

Optimisation des performances temporelles de circuits intégrés VLSI

Thèse soutenue publiquement le 18/03/1998 devant le jury:

Président :	**A. Kheladi**	**Professeur USTHB**
Rapporteur :	**A. Farah**	**Maître de conférences ENP**
Examinateur :	**M. Haddadi**	**Maître de conférences ENP**
Examinateur :	**M. Selmane**	**Professeur USTHB**
Examinateur :	**M. Mehenni**	**Maître de conférences ENP**

Résumé

Les travaux que nous avons entrepris sont tous liés à un aspect particulier de la conception assistée par ordinateur de circuits intégrés VLSI (Very Large Scale Integration). Cet aspect est l'estimation et l'optimisation des performances temporelles des circuits VLSI. L'importance du facteur temps pour les circuits intégrés est telle que certains experts l'ont qualifiée de troisième dimension du silicium. Les motivations de ces travaux sont liées à l'évolution des circuits intégrés. En effet, les marchés de circuits intégrés (particulièrement les unités de traitement) exigent des circuits de plus en plus rapides. De plus, la taille de ces circuits augmente continuellement (elle atteint la taille de plusieurs millions de transistors) rendant, de ce fait, la mesure et l'optimisation des performances de plus en plus difficiles. De plus avec les dernières évolutions de la technologie notamment les technologies submicroniques, les performances des circuits intégrés ont tendance à être dégradées. Ce qui explique l'impérieuse nécessité de développer des outils CAO spécialisés, qui ont pour tâche d'assister les concepteurs de circuits intégrés durant la phase d'estimation et d'optimisation des circuits. En outre, compte tenu de l'importance de cette tâche, il est nécessaire de considérer les circuits intégrés à différents niveaux de représentation Dans cette thèse sont développes plus particulièrement des algorithmes d'optimisation au niveau physique, et ce d'abord au niveau porte et ensuite au niveau transistor.

Dédicaces

à mes parents

à mon épouse

à mes enfants

Remerciements

Je remercie Monsieur H. Bessalah Directeur du CDTA, pour son aide et pour le support materiel mis à ma disposition durant la préparation de cette thèse.

Je remercie Monsieur A. Farah Maître de conférences à l'ENP, pour le suivi attentif des travaux de cette thèse et pour sa bienveillante disponibilité.

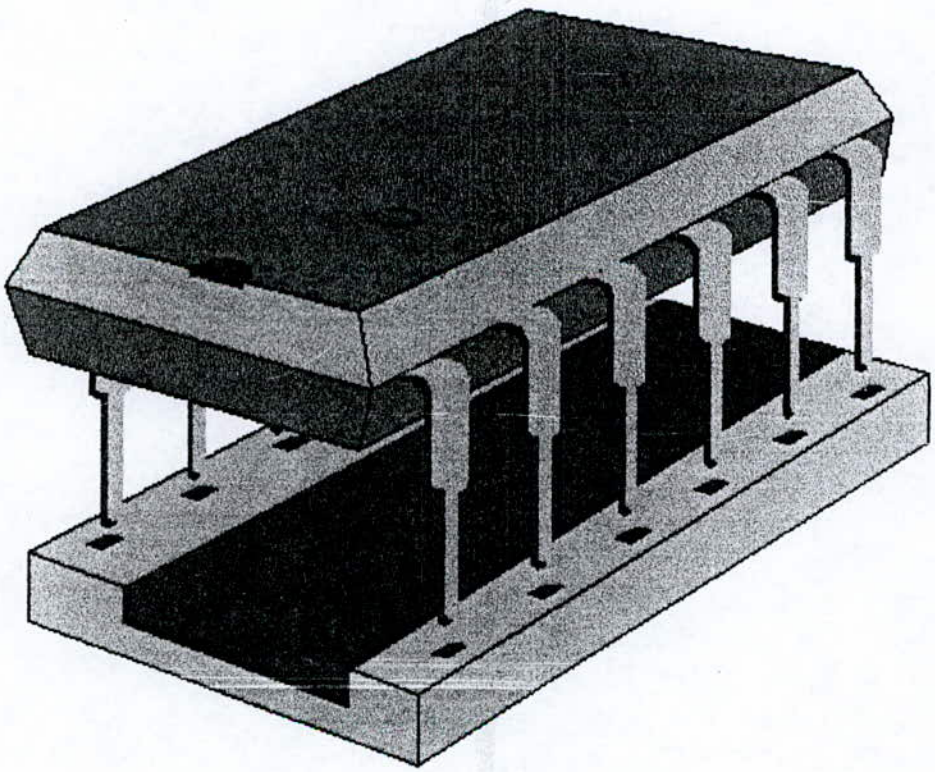
Je remercie les membres du jury qui m'ont fait l'honneur d'accorder de l'intérêt à mes travaux.

Je remercie tous mes collègues du Laboratoire de Micro-électronique du CDTA pour les fructueuses discussions que nous avons eues.

Table des matières

CHAPITRE I

INTRODUCTION



Introduction

Le besoin d'ordinateurs ou d'unités de traitement de plus en plus rapides est exprimé chaque jour par la recherche dans des domaines tels que la météorologie, la sismologie, la recherche spatiale , la modélisation du cerveau humain... Il est également exprimé par l'industrie, telles que l'industrie aéronautique, automobile etc. ... Le facteur commun à tous les secteurs demandeurs est le besoin de puissance de calcul de plus en plus grande pour étudier et modéliser avec de plus en plus de précision, des phénomènes de plus en plus complexes.

Ce problème a été pris en charge par la recherche et cela à des niveaux variés. On peut citer principalement :

- d'abord, au niveau architectural puisque des architectures non Von-Neuman ont vu le jour (vectorielle, pipe-line multiprocesseur, etc...) ainsi qu'au niveau des arithmétiques.
- au niveau algorithmique où des algorithmes parallèles ont été développés et que des systèmes d'exploitation multitâches multiprocesseurs sont en train d'émerger.
- Au niveau circuit où des techniques de conception nouvelles ont été développées pour permettre de concevoir des circuits de plus en plus rapides
- Au niveau des technologies des semi-conducteurs où des technologies de plus en plus rapides ont été développées (AsGa, HCMOS, BiCMOS, etc...). Toutefois, le facteur le plus décisif semble être l'intégration des circuits sur silicium. Grâce à elle des ordinateurs entiers ont pu être intégrés sur des puces (microprocesseurs), ce qui a permis d'améliorer

sensiblement les performances des ordinateurs en terme de vitesse tout en réduisant les coûts par rapport aux techniques traditionnelles organisées autour du circuit imprimé.

Grâce à l'évolution de la technologie, les dimensions des circuits n'ont cessé d'être réduites. Ce phénomène peut globalement être caractérisé par ce qu'on appelle le facteur d'échelle λ (généralement exprimé en microns). Ce dernier représente la dimension du plus petit motif réalisable sur circuit intégré. Pour quantifier les progrès réalisés, on peut dire que λ est passé en vingt ans de quelques dizaines de microns à quelques dixièmes de microns. Les conséquences sur les performances en terme de vitesse ont été importantes puisque la réduction du facteur d'échelle a permis d'augmenter la vitesse des porteurs tout en diminuant les distances parcourues par eux.

Toutefois, cette tendance semble se ralentir avec l'avènement ces dernières années des technologies submicroniques, compte tenu de l'apparition de phénomènes physiques inhérents aux dimensions submicroniques [3], tels que la saturation de la vitesse des porteurs, les effets des champs électriques, le phénomène des électrons chauds etc. ... D'autre part, la réduction du facteur d'échelle a permis d'obtenir des densités d'intégration de plusieurs millions de transistors. Par conséquent, des problèmes sérieux de consommation d'énergie ont fait leur apparition. Ces derniers peuvent même se manifester sous forme de dissipation de chaleur qui peut même provoquer un fonctionnement incorrect des circuits. L'une des solutions proposées à ce problème est la nécessité de réduire les tensions d'alimentation qui sont en train de descendre en dessous de 5 Volts. Ceci, malheureusement, a pour effet la dégradation des performances temporelles des circuits.

La conjonction des besoins croissants en matière de circuits rapides, de l'augmentation exponentielle de la taille des circuits avec le temps et de l'évolution de la technologie avec ses effets non désirés sur les performances, ont fait ressentir le besoin de méthodologies efficaces pour l'optimisation automatique des performances en terme de vitesse de circuits intégrés. Cette optimisation s'effectue sur la base de spécifications exprimées par les concepteurs ainsi que de contraintes sur les caractéristiques du circuit désirées telles que ses dimensions et par conséquent son coût, sa consommation etc. ....

Traditionnellement cette tache était effectuée par les concepteurs qui , en fonction de leur expérience optimisaient leurs circuits de façon empirique. Ce qui n'était pas sans conséquences sur la qualité de ces derniers. De plus, il était nécessaire de passer à l'étape de la fabrication des circuits pour mesurer les performances ainsi obtenues. Ce qui avait pour conséquences d'augmenter le coût et les délais du projet, étant donné le coût élevé et le temps d'attente important nécessaire à la fabrication d'un circuit intégré. Ces limitations ont motivé le développement d'outils informatiques de conception assistée par ordinateur (CAO) qui effectuent automatiquement l'optimisation de circuits (voir figure 1.1).

Les difficultés rencontrées par ces outils sont liées à la grande diversité des circuits intégrés. En effet, ceux-ci possèdent plusieurs niveaux de représentation au sein desquels plusieurs configurations sont possibles, chacune possédant un nombre élevé de paramètres. Par conséquent, l'espace des solutions est quasiment infini pour un circuit donné. Comme la taille de ces circuits est très grande et ne cesse d'augmenter chaque jour, il en résulte que les problèmes d'optimisation de circuits sont généralement considérés comme difficiles et appartiennent , pour la plupart, à la classe des problèmes NP-difficiles [9].

Deux caractéristiques fondamentales des algorithmes d'optimisation de circuits intégrés devront être considérées :

- la performance des algorithmes
- la précision des résultats

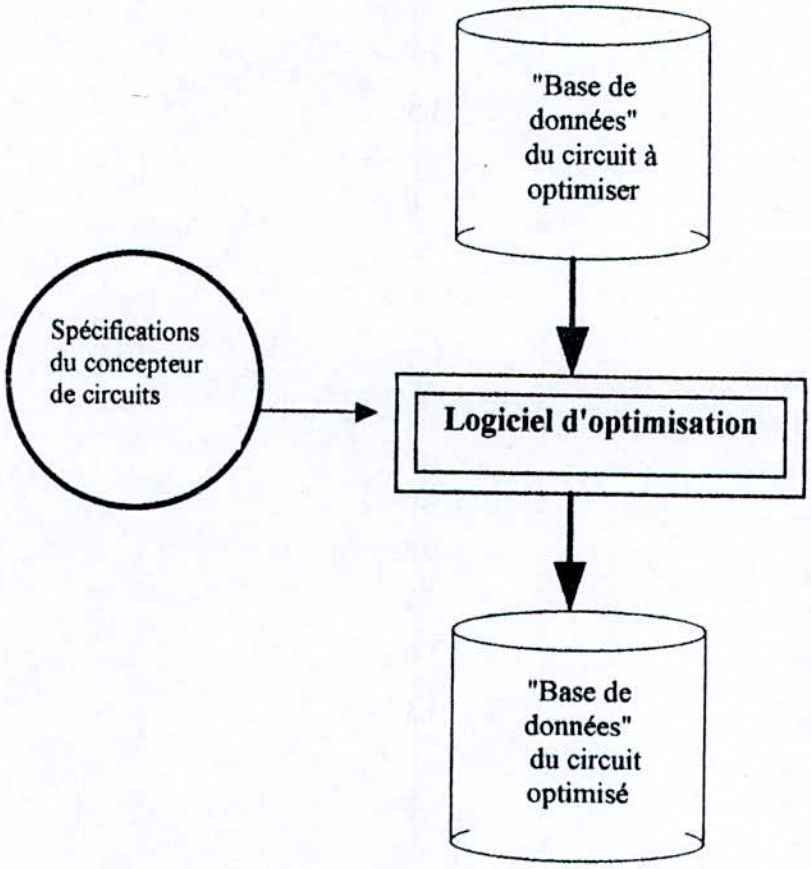


Figure 1.1 Block diagramme général d'un logiciel d'optimisation

Ces deux caractéristiques sont antagonistes, puisque plus la qualité des circuits optimisés sera élevée plus les temps de calcul seront importants. Par contre, plus un algorithme sera

rapide plus la qualité des circuits optimisés sera faible. Par conséquent, le compromis vitesse/précision sera un facteur déterminant pour le choix des algorithmes.

Dans le cadre de cette thèse nous aborderons le problème de l'optimisation de circuits intégrés digitaux au niveau du chapitre 2, en présentant un rappel des éléments de base pour les CI, la théorie des graphes, l'optimisation. De plus la problématique de l'optimisation au niveau physique sera présentée, puis placée dans le contexte du processus global de conception.

Au chapitre 3, le problème de l'optimisation de circuits sera d'abord introduit par la description de la problématique au niveau portes logiques, ainsi qu'une méthode de résolution dans le cas d'une technologie mixte appelée BiCMOS. Notre contribution dans ce domaine est le développement d'un outil CAO d'optimisation du temps de réponse pour les circuits CMOS/BiCMOS dont les résultats ont fait l'objet d'une publication [35] (voir annexe 3).

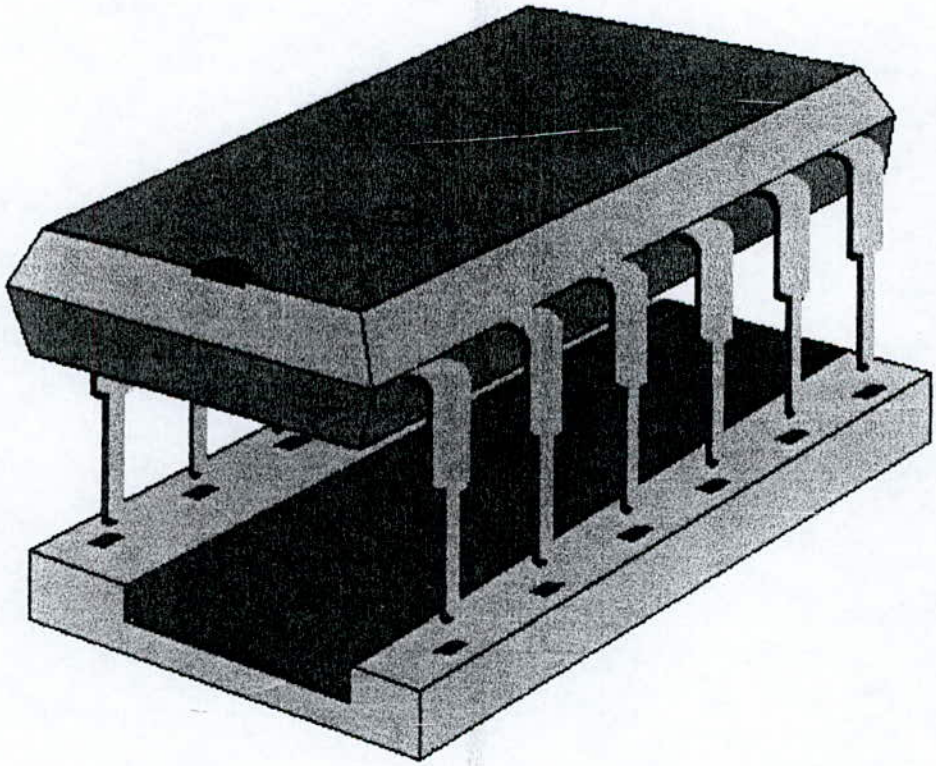
Pour une optimisation de qualité, un raffinement à une échelle plus petite, celle du niveau transistor est parfois nécessaire. Le chapitre 4, est consacré à cette problématique. Nous avons développé, dans ce domaine, une méthodologie qui s'appuie sur la résolution d'un problème sous-jacent au problème d'optimisation, appelé problème de détermination des sens de flux des transistors. Les résultats de la validation expérimentale de la méthode que nous avons proposée ont été publiés [33] (voir annexe 1) tandis qu'une démonstration formelle de la justesse de l'algorithme développé est en voie de publication [36] (voir annexe 3).

Cette étude sera achevée par une présentation des conclusions des travaux présentés, ainsi que des suggestions de travaux futurs qui devront aboutir au développement d'une méthodologie

unifiée et hiérarchique d'optimisation de circuits intégrés VLSI comme perspectives à long terme.

Enfin, pour éviter de reprendre certains longs développements et démonstrations faisant partie de nos travaux, qui ont déjà fait l'objet de publications (ou en voie de l'être), les articles en question ont été rajoutés en annexe.

CHAPITRE II



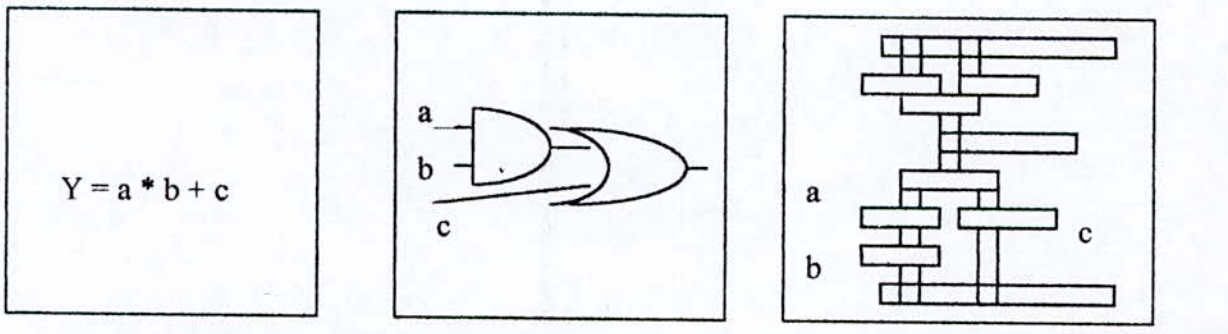
Optimisation temporelle de circuits intégrés digitaux

1 Introduction aux circuits intégrés (CI)

Un CI est un circuit électronique entièrement réalisé sur un substrat semi-conducteur comme le silicium. Les composants élémentaires des circuits intégrés digitaux sont les transistors interconnectés entre eux par des niveaux de métallisation. La concrétisation d'un circuit intégré nécessite 4 grandes étapes:

- **La conception** qui consiste principalement en la modélisation, la synthèse, l'optimisation et la validation.
- **La fabrication** qui consiste principalement en la fabrication des masques et la fabrication des tranches de silicium.
- **Le test**
- **Le packaging** qui consiste principalement en la découpe des tranches de silicium et l'encapsulation.

L'étape de conception aboutit à une description d'un CI (ou puce) appelée dessin de masques ou layout. Un CI est généralement représenté par trois vues appelées aussi domaines de description, on distingue [1] :



a) domaine comportemental b) domaine structurel c) domaine physique ou géometrique

Fig 2.1 Illustration des trois domaines de représentation

- **Le domaine comportemental :** où le circuit est représenté par exemple sous la forme d'équations logiques en langage évolué. Dans ce domaine, on s'intéresse à ce que fait le circuit sans s'occuper des détails de réalisation. Un circuit est considéré comme une boite noire ayant une interface (des entrées et des sorties), ainsi que des fonctions qui décrivent le comportement des sorties en fonction des entrées.
- **Le domaine physique :** où le circuit est représenté sous forme de layout, de floor-plan , etc. Dans ce domaine on s'intéresse à la façon avec laquelle est réalisé le circuit, en faisant abstraction de la fonctionnalité de ce dernier.
- **Le domaine structurel :** où le circuit est représenté par exemple sous forme de transistors, ports logiques, blocs fonctionnels . Ce domaine représente un domaine intermédiaire entre le domaine comportemental et physique. Dans ce domaine on s'intéresse plus particulièrement à la structure du circuit. Il est à noter que cette structure est décrite en termes de composants et de connections.

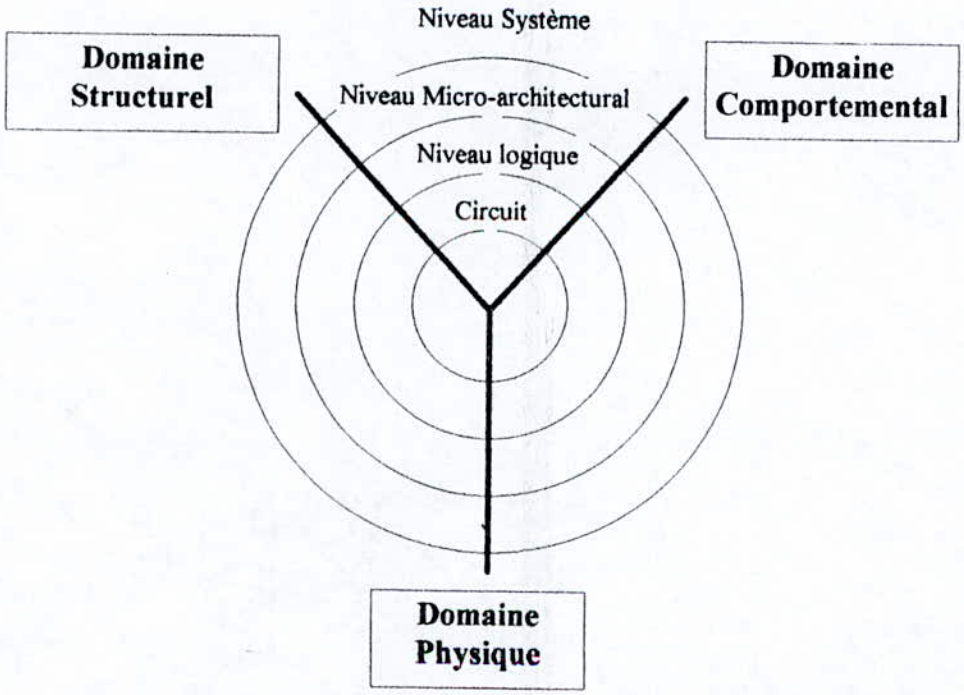


Figure 2.2 "Y" de gajski

Chaque domaine est divisé en niveaux de description, on distingue généralement les niveaux suivants [2]: le niveau système, le niveau micro-architectural, le niveau logique et le niveau circuit. Ce partitionnement permet de décomposer le processus de conception de CI en niveaux d'abstraction indépendants, ce qui a pour conséquence de faciliter ce processus.

L'importance du domaine physique, au point de vue performances est fondamentale, puisque celles-ci sont déterminées par les caractéristiques des représentations des CI dans ce domaine. En effet, les dimensions des composants élémentaires vont déterminer la surface du circuit mais aussi le courant qui passe à travers eux et par conséquent vont déterminer les délais de propagation ainsi que l'énergie dissipée. De plus, la longueur des interconnexions qui n'est connue que lorsque le circuit est réalisé et donc décrit au niveau du domaine physique, est très importante puisqu'elle va déterminer les capacités d'interconnexion et donc les charges attaquées par les composants. Ces charges ont un effet déterminant sur la vitesse de fonctionnement du circuit. Ceci est d'autant plus vrai qu'avec l'avènement des technologies submicroniques le délai dû aux interconnexions est devenu dominant par rapport à celui dû aux composants.

2 Méthodologies de conception et outils CAO

Il existe plusieurs méthodologies de conception de CI. Parmi les plus utilisées on distingue:

La méthodologie Full-custom (ou sur mesure) : Cette méthodologie se distingue par le fait qu'un circuit est partitionné en une hiérarchie de sous-circuits fonctionnels. Il est à noter que les sous-circuits de base (les sous-circuits terminaux de la hiérarchie) sont développés en termes de transistors. Un CI est organisé en termes de blocs fonctionnels pouvant avoir des dimensions arbitraires ainsi qu'une position au sein de la puce qui peut être, elle aussi,

arbitraire. L'espace non occupé par les blocs est utilisé par les interconnections. Par conséquent, ce style offre une grande liberté pour les concepteurs et permet d'obtenir les circuits les plus performants et aussi les plus compacts au prix cependant d'un temps de conception très élevé.

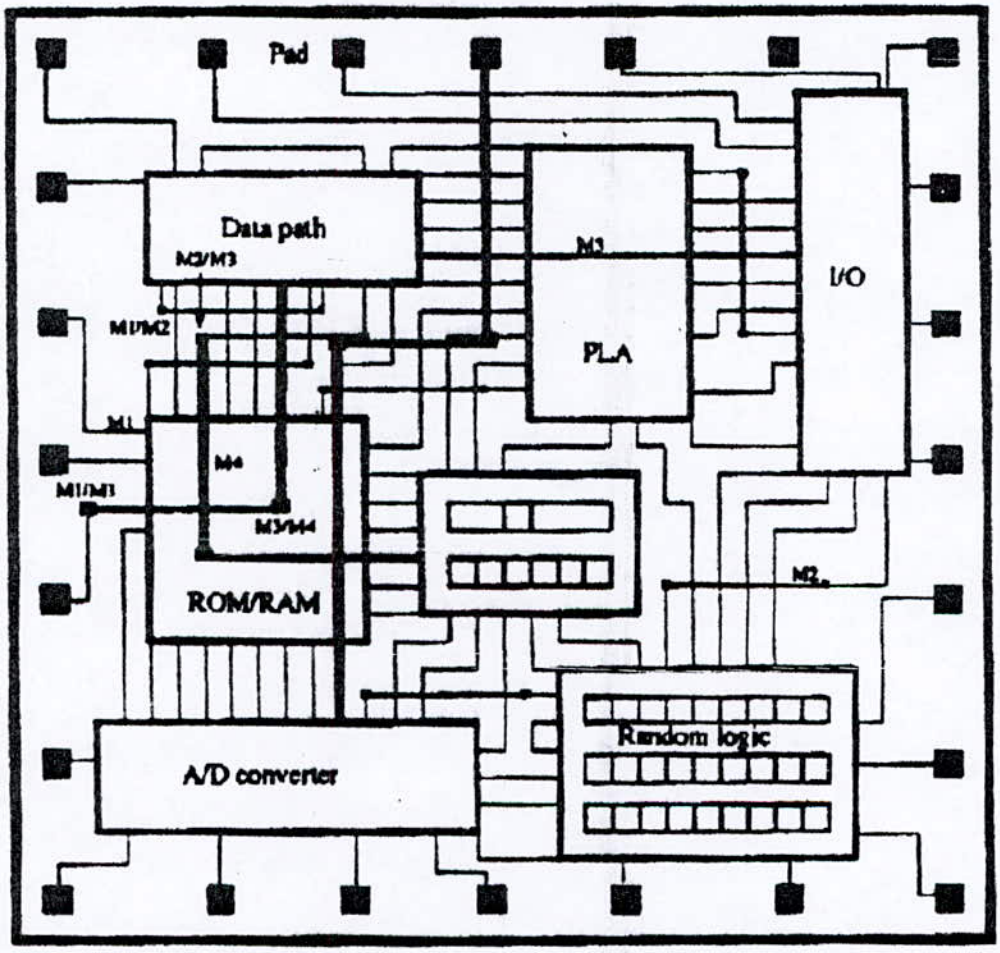


Fig 2.3 Exemple de layout full-custom

La méthodologie standard-cells (cellules standards): Cette méthodologie est considérée comme une méthodologie semi-custom. Elle se distingue par le fait que les dessins de masques sont non hiérarchiques mais plutôt constitués d'un assemblage de composants rectangulaires de même hauteur appelés cellules. Celles-ci sont organisées par rangées de cellules collées les unes aux autres. L'espace réservé au routage des interconnexions est constitué de l'espace existant entre les rangées de cellules. La conception de CI à l'aide de cette méthodologie se fait à base de composants choisis dans une bibliothèque de cellules. Il est à noter que cette méthodologie produit des circuits qui , par rapport à la méthodologie full-custom, consomme plus de surface. Toutefois le temps de conception est beaucoup plus court.

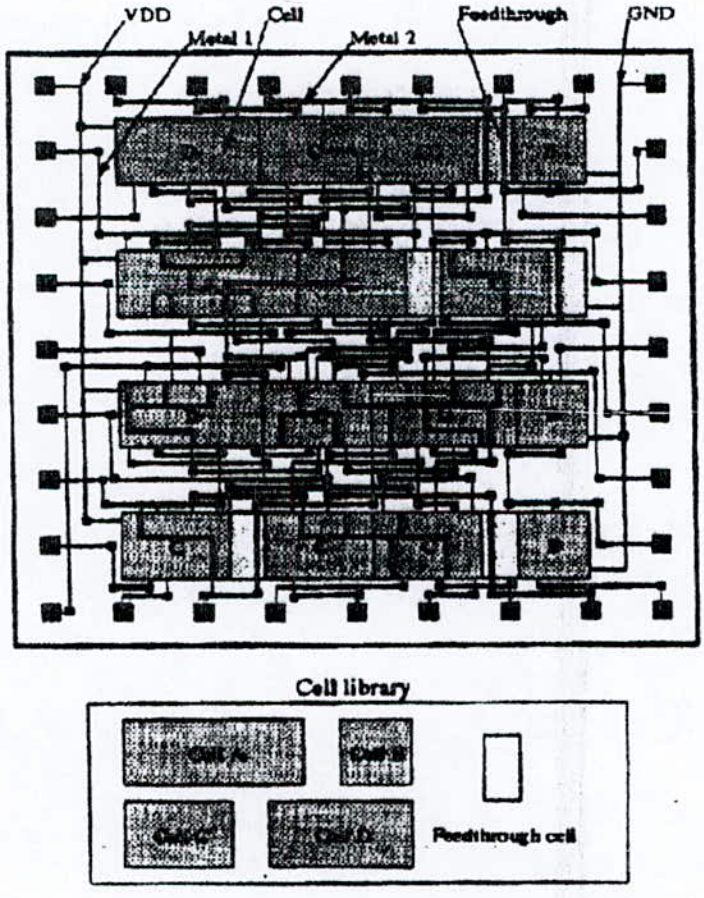


Fig 2.4 Exemple de layout standard-cells

Les niveaux de description des circuits sont en fait, intimement liés à la méthodologie de conception. En effet, pour la méthodologie full-custom (manuelle), la conception d'un circuit passera par les trois domaines (i.e. comportemental, structurel et physique), tandis que pour la méthodologie semi-custom (semi-automatisée), seules les deux premiers domaines (comportemental et structurel) seront utilisés. Enfin, pour la méthodologie dite design automation (automatisée) , la conception concernera uniquement le premier domaine (comportemental).

Il est à noter que toute méthodologie est associée à des outils informatiques (CAO). Généralement on distingue deux catégories d'outils associés à chaque niveau d'abstraction:

- les outils d'aide à la conception,
- les outils de vérification et d'analyse.

Les premiers assistent le concepteur durant la conception proprement dite du circuit en lui fournissant des outils informatiques (éditeurs, etc...) ou automatiques (générateurs, compilateurs, etc...). Les seconds permettent de vérifier si le circuit conçu ne contient pas d'erreurs et est conforme au cahier des charges (simulateurs, analyseurs, etc...).

3 Les principaux outils de CAO

3.1 Les outils au niveau comportemental.

Au niveau comportemental, le circuit est généralement décrit à l'aide de langages évolués tels que des langages de programmation, langages de description de machines d'états finis, langages de description d'équations booléennes, etc... Ces descriptions seront vérifiées par des outils de simulation de haut niveau. Dans le cas de la méthodologie de conception automatisée, le concepteur dispose de logiciels tels que les outils de synthèse de haut niveau

ou les compilateurs de silicium. Ceux ci acceptent en entrée une description comportementale ainsi qu'un ensemble de spécifications relatives à la surface, la vitesse, la consommation, etc..., pour fournir en sortie directement le dessin de masques.

3.2 Les outils du domaine structurel.

Au niveau structurel, le concepteur dispose d'outils de description généralement graphiques qui lui permettent de décrire un circuit sous des formes diverses (registres, portes, transistors ...). Une fois le circuit décrit, le concepteur utilise une batterie de simulateurs et d'analyseurs pour valider le circuit et pour l'analyser en termes de performances, de consommation, etc... . A ce stade , le concepteur dispose d'outils de génération automatique de structures régulières qui fournissent le layout de circuit moyennant l'indication du type de circuit et ses paramètres. Dans le cas de la méthodologie semi-automatisée, pour réaliser le circuit proprement dit, un ensemble d'outils de placement et de routage sont disponibles pour agencer et connecter les différents blocs de layout.

3.3 Les outils du domaine physique.

Au niveau physique, le concepteur dispose d'éditeurs graphiques évolués qui permettent de gérer la grande masse de rectangles du layout et leur organisation hiérarchique tout en offrant des options de visualisation sophistiquées. Une fois le layout conçu entièrement, le concepteur dispose d'outils de vérification physique DRC (Design Rule Checker) qui permettent de vérifier que les règles technologiques d'agencement du layout sont respectées. Enfin, un outil d'extraction permet d'extraire à partir du layout une description à base de transistors qui pourra être utilisée par un simulateur électrique pour effectuer des simulations analogiques de circuits.

3.4 Benchmarks et vérification expérimentale des résultats

Dans le domaine de la CAO/VLSI lorsqu'un logiciel est conçu, il est important d'évaluer ses performances en termes de vitesse d'exécution et de qualité de traitements. Pour cela, il est nécessaire de comparer ces performances par rapport à celles d'autres logiciels effectuant les mêmes taches. Il est alors indispensable de comparer ces performances sur des circuits de tests qui sont standardisés, pour permettre leur disponibilité à toute personne désirant tester son logiciel. Dans ce cadre, il existe généralement pour chaque domaine de recherche de la CAO/VLSI, un ensemble de circuits appelés benchmarks qui sont proposés comme jeux d'essai ou étalon de mesure, pour le test et l'évaluation de tout logiciel. Parmi les plus connus on peut citer:

- Les benchmarks combinatoires ISCAS'85 (International Symposium on Circuits And Systems) qui sont utilisés initialement dans le domaine du test, et qui sont maintenant couramment utilisés dans les domaines de la synthèse logique, de l'optimisation etc.
- Les benchmarks ISCAS'90 qui sont constitués de circuits séquentiels.
- Les benchmarks MCNC (Microelectronics Center of North Carolina).

4 Représentation des circuits intégrés.

Aujourd'hui la conception d'un CI n'est possible qu'avec des logiciels de CAO . Ces derniers assistent les concepteurs en leur permettant de visualiser leurs circuits, de les simuler, de les vérifier, etc... et même de les concevoir automatiquement grâce aux outils de synthèse. Pour cela, une représentation en mémoire centrale des CI est nécessaire, sous forme de structures de données qui permettent à des programmes d'effectuer les traitements désirés sur les circuits. En réalité, il existe plusieurs représentations possibles en fonction du domaine de représentation, et aussi en fonction du type d'application envisagée. Le dénominateur

commun à ces structures est qu'un CI est généralement représenté sous forme d'un graphe en mémoire centrale.

Parmi les représentations les plus répandues, on peut citer :

- le CDFG (control data flow graph),
- le BDD (binary decision diagram),
- le réseau booléen (boolean network) ,
- le netlist logique,
- le channel graph,
- le neighborhood layout graph,

5 Rappels et terminologie des graphes.

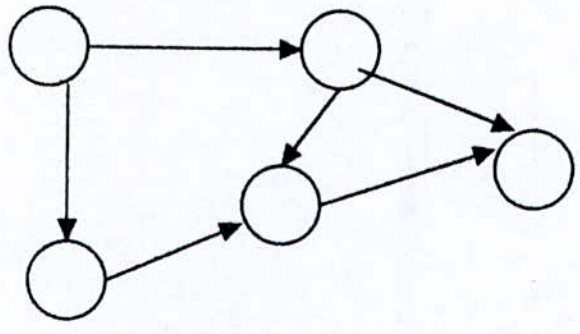


Figure 2.5 Exemple de graphe

Un graphe G(V,E) [7] est constitué d'un ensemble V et d'une relation binaire sur V. Les éléments de V sont appelés sommets alors que les éléments de E sont appelés arêtes. Une arête est dénotée par un couple de sommets nommés extrémités. Une arête est dite incidente à un sommet si ce dernier est une des deux extrémités de cette arête. Le degré d'un sommet est le nombre d'arêtes incidentes à ce sommet. Deux sommets sont dits adjacents s'il existe une arête les liant. Dans ce cas, on dit que cette arête les connecte. On parle de graphe valué si les arêtes et/ou les sommets possèdent un poids (valeur).

5.1 Graphes non-orientés.

Dans un graphe non-orienté, les arêtes sont constituées d'une paire non ordonnée de sommets. On dit qu'un sommet est adjacent à un autre quand il existe une arête incidente à ces deux sommets. Un chemin est une séquence d'arêtes distinctes. Un cycle est une séquence d'arêtes fermées. Un graphe est dit connexe si toute paire de sommets est liée par un chemin. Un graphe sans cycle est dit acyclique. Un arbre est un graphe connexe acyclique.

5.2 Graphes orientés.

Un graphe orienté est défini de la même façon qu'un graphe non-orienté, à la différence que les arêtes (appelées arcs) sont définies par un couple orienté de sommets. Ces sommets sont appelés dans l'ordre, sommet initial et sommet terminal. Un chemin orienté est une séquence d'arcs tels que le sommet terminal de chaque arc correspond avec le sommet initial de l'arc précédent dans la séquence. On appelle degré entrant (respectivement degré sortant) d'un sommet, le nombre d'arcs ayant ce sommet comme extrémité terminale (respectivement initial).

5.3 Algorithmes pour les graphes.

On appelle exploration d'un graphe tout procédé déterministe qui , durant l'examen exhaustif des sommets d'un graphe, permet de choisir, à partir du dernier sommet visité, le sommet suivant. Les explorations (search) sont à la base de la majorité des algorithmes traitant des graphes. On en distingue généralement deux :

5.3.1 L'exploration en profondeur ou DFS (depth-first-search).

Dans ce type d'exploration le graphe est exploré sommet après sommet en choisissant à chaque fois , à partir de chaque sommet, un arc incident et en continuant l'exploration à partir du sommet adjacent. Lorsque l'algorithme a fini d'explorer les arcs incidents d'un sommet, il revient au sommet précédent pour considérer des arcs non encore explorés.

Il est à noter que ce mode d'exploration utilise une pile pour mémoriser les sommets restant à explorer.

5.3.2 L'exploration en largeur ou BFS (breath-first-search).

L'idée de base est d'explorer tous les sommets adjacents d'un sommet donné avant d'en explorer d'autres. En d'autres termes, si l'exploration commence à partir d'un sommet donné, l'algorithme détermine d'abord tous les sommets qui lui sont adjacents. Puis, après les avoir explorés, il détermine et explore, à leur tour, tous les sommets qui leur sont adjacents, et ainsi de suite jusqu'à ce que tous les sommets du graphe soient explorés. Il est à noter que durant ce processus , les sommets en attente d'être explorés sont mémorisés dans une file.

6 Rappel de la définition d'un problème d'optimisation.

D'une façon générale, un problème d'optimisation est défini comme suit [6]. Il s'agit de trouver x appartenant à R^n pour :

minimiser f(x) (appelée fonction objectif)

avec les contraintes $g_i(x) >= 0 \qquad i=1,n$

$$h_j(x) = 0 \qquad j = 1,p$$

où f, g_i, h_j **sont des fonctions de x appartenant à** R^n

On distingue deux catégories de problèmes d'optimisation en fonction de la nature de x qui peut être continue ou discrète. Dans ce dernier cas, on parle d'optimisation combinatoire.

7 Définition de la performance d'un circuit.

Elle peut être définie différemment selon le niveau de description ou le type de circuit [1].

Circuit combinatoire : La performance d'un circuit combinatoire est égale au délai de propagation à partir des entrées vers les sorties. Ce dernier est défini comme étant le temps nécessaire pour propager jusqu'aux sorties, l'effet d'une modification des tensions présentes aux entrées d'un circuit.

Circuit séquentiel : La performance d'un circuit séquentiel synchrone est égale à son temps de cycle, c'est-à-dire, la période minimale de l'horloge pouvant être utilisée pour synchroniser ce circuit. Il est à noter que le délai à travers les composants combinatoires d'un circuit séquentiel représente une borne inférieure du temps de cycle .

Circuit décrit au niveau architectural : La performance d'un circuit décrit au niveau architectural (séquence d'opérateurs utilisant une implémentation séquentielle synchrone) est égale à sa latence elle-même égale au nombre de cycles d'horloge nécessaires pour exécuter les opérations. Dans le cas d'une architecture pipe-line, la performance est mesurée par le débit (troughput) qui est égal à la cadence à laquelle les données sont traitées.

8 Définition de l'optimisation de performance de circuits.

L'optimisation consiste à :

- minimiser le délai de propagation pour les circuits combinatoires,
- minimiser le temps de cycle et la latence pour les circuits séquentiels,
- maximiser le débit pour les circuits pipe-line.

Si on se réfère à la formulation introduite au paragraphe 6, la fonction objectif f(x) représente pour chaque cas: le délai de propagation, le temps de cycle, la latence et enfin le débit. Tandis que les fonctions gi(x), hj(x) représentent des contraintes sur des paramètres tels que la surface, la consommation d'énergie fixés par l'utilisateur. Ces contraintes peuvent être aussi d'ordre technologiques, telles que celles des dimensions minimales des transistors autorisées pour une technologie de fabrication donnée.

Nous nous intéressons dans notre étude, plus particulièrement, à l'optimisation des circuits combinatoires, étant donné que les méthodes développées peuvent être généralisées aux circuits séquentiels. Ceci peut être fait en considérant séparément chaque bloc combinatoire des circuits séquentiels traités , en faisant abstraction des registres pour y appliquer les algorithmes d'optimisation spécifiques aux circuits combinatoires.

Le problème de l'optimisation de performance d'un circuit peut être vu de manière plus formelle comme suit : Les différentes implémentations possibles d'un circuit définissent un espace de conception (design space) qui est constitué d'un ensemble de points (design point) représentant chacun une implémentation possible du circuit. A chaque point ou implémentation sera associé un ensemble de valeurs représentant les performances. L'optimisation de circuits consistera à rechercher le "meilleur" point de l'espace de conception ou configuration de circuit qui optimise les objectifs.

La résolution de ce problème est difficile compte tenu de:

- la taille des circuits qui peut être très importante.
- la variété des technologies de circuits VLSI (CMOS, BiCMOS, ECL) et des méthodologies (full custom, standard cells, gate arrays, FPGA) qui nécessitent chacune des traitements spécifiques.
- la difficulté d'estimer efficacement et précisément les temps de propagation dans les circuits intégrés.

D'autres difficultés sont elles liées à la conception des circuits:

- les différents styles et techniques utilisés par les concepteurs de circuits (statiques ou dynamiques, combinatoires ou séquentiels etc.) qu'il faut considérer.
- les différents domaines et niveaux de représentation de circuits qui nécessitent chacun des algorithmes spécifiques.

9 Les principaux paramètres utilisés durant l'optimisation

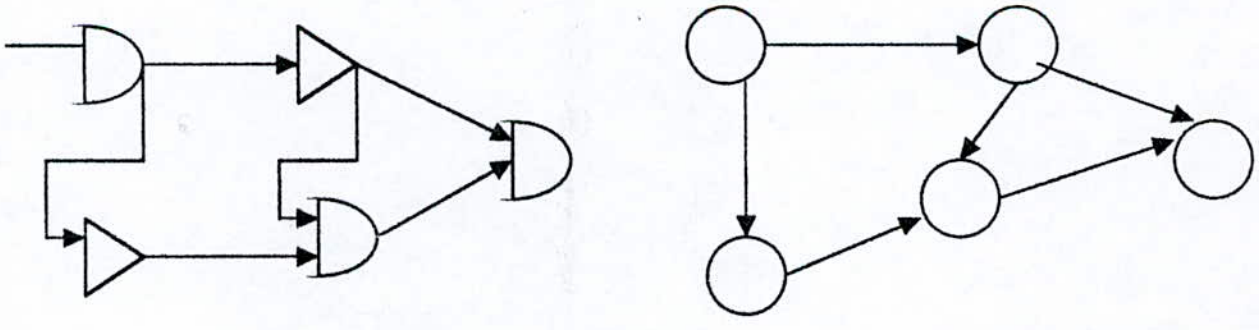


Fig 2.6 Représentation d'un circuit par un graphe

9.1 La surface de silicium:

Son importance est due au fait qu'elle détermine le coût de fabrication, puisque celui-ci est proportionnel à la surface utilisée. La surface est le paramètre qui , relativement est le plus facile à mesurer. En effet, elle est égale à la somme des surfaces des composants ajoutée à la surface occupée par les interconnexions. La surface A est généralement estimée à l'aide de la formule suivante:

$$A = \sum_i a_i$$

(a_i surface du i ^{eme} composant élémentaire)

Cette estimation ne considère pas la surface d'interconnexion, car cette dernière ne varie pas avec les méthodes d'optimisations basées sur le dimensionnement.

9.2 La consommation d'énergie:

L'énergie dépensée sera fonction des courants électriques réellement dissipés. Les charges vont essentiellement se déplacer durant les transitions (consommation dynamique). Aussi l'énergie sera fonction du nombre de transitions qui vont avoir lieu dans un circuit. Ces transitions sont calculées en termes de probabilités de transitions qui sont calculées à l'aide de modèles probabilistes [17] tel que:

$$P_{av} = 0.5 / Tc \; V_{DD}^{\,2} \sum C_i \; P_t(x_i).$$

ou **xi représente les noeuds du circuit**
Tc la période d'horloge
Ci capacité du noeud xi

9.3 Le délai de propagation:

Ce dernier est plus délicat à estimer car contrairement à la surface et à la puissance, c'est un paramètre non additif. C'est-à-dire qu'on ne le calcule pas comme étant la somme de tous les délais du circuit. Par contre il va dépendre à la fois de la structure et du comportement du circuit analysé. En effet, le délai de propagation d'un circuit est égal à la somme des délais des composants à travers le chemin de propagation le plus lent (appelé chemin critique). On associe à chaque noeud du circuit (y compris les sorties) une mesure appelée temps-arrivée (arrival time) qui est le temps où la donnée sur ce noeud est stable et par conséquent utilisable.

$$t_i = d_i + \max t_j$$

ou j tel que $(v_j, v_i) \in E$
t_i temps arrivée de la porte i
d_i délai de propagation de la porte i
E : ensemble des arêtes du graphe modélisant le circuit

Dans le cadre de l'optimisation, on associe aussi aux portes le temps requis pour atteindre les spécifications fixées par l'utilisateur. Ce temps est défini comme suit:

$$t'_i = \min t'_j - d_j$$

ou j tel que $(v_i, v_j) \in E$
t_i', t_j', d_j définis comme précédemment

Grâce au temps d'arrivée et au temps requis, on peut calculer une mesure de flexibilité pour chaque noeud vi de l'ensemble V des noeuds du graphe, appelée "slack time" qui est égale à :

$$s_i = t'_i - t_i \quad \forall v_i \in V$$

Le délai dépend d'un nombre important de facteurs tels que: les caractéristiques du semi-conducteur comme le dopage etc, les caractéristiques du circuit tels que les charges, les dimensions des composants etc., et enfin les phénomènes liés aux signaux tels que les vecteurs de tests, la forme des signaux, leur localisation etc.

Il est à noter que dans le cadre de cette étude nous nous concentrerons essentiellement sur les paramètres délai et surface qui sont les paramètres généralement étudiés dans la littérature [20] [21] [18] [23] [24] etc. (l'extension au paramètre puissance est généralement aisée).

10 Estimation des performances temporelles

Traditionnellement les performances des CI étaient déterminées à l'aide de simulations durant lesquelles l'utilisateur simulait systématiquement toutes les combinaisons de signaux. Cependant cette approche est devenue avec le temps infaisable étant donné la taille croissante des circuits et par conséquent le nombre prohibitif de vecteurs d'entrée . Une solution à ce problème a consisté dans un premier temps à éviter de faire une analyse exhaustive des circuits mais plutôt à choisir un nombre réduit de combinaisons de données et à les simuler. Toutefois, cette approche a pour inconvénient de dépendre du choix des données, sans que rien ne garantisse la précision des résultats. Cette situation a motivé le développement d'outils CAO appelés analyseurs temporels. Ces derniers sont capables d'évaluer les performances temporelles de CI sans avoir recours à la simulation en effectuant une analyse indépendante de données (pattern-independent-analysis). L'approche utilisée consiste à rechercher le chemin de propagation le plus lent en considérant systématiquement les cas de propagation les plus défavorables (worst-case analysis). Ce chemin est appelé chemin critique (critical path) [8].

10.1 Modélisation du temps de propagation

La difficulté liée à l'estimation du temps de propagation vient du fait qu'elle dépend de plusieurs facteurs. On peut citer les plus importants tels que les dimensions des transistors, les charges attaquées (qui dépendent des interconnexions et aussi des charges parasites des transistors) et de la nature et de la forme des signaux. Dans tous les cas de figures, il s'agira toujours de faire des estimations dans les cas les plus défavorables car :

- l'analyse étant indépendante des données, tous les paramètres ne sont pas connus pour permettre une estimation précise;
- une surestimation du délai aura pour conséquence une perte relative des ressources telles que la surface. Par contre une sous-estimation conduirait à un fonctionnement incorrect du circuit ; ce qui n'est pas acceptable.

10.2 Le chemin critique.

Partant des entrées d'un circuit combinatoire, les signaux traversent , en un temps égal au délai de propagation, les composants qui sont liés aux entrées. Ensuite, ils traversent les composants eux-mêmes liés aux composants déjà traversés. Ce processus se poursuit ainsi jusqu'à atteindre les sortie du circuit. Ce faisant, des chemins de propagation des signaux sont déterminés. On associe à chaque chemin un délai qui est égal à la somme des délais de propagation des composants qui constituent ce chemin. Parmi tous les chemins possibles, le chemin le plus lent, appelé chemin critique , va déterminer le délai de tout le circuit. Ce temps est égal au délai de propagation du chemin le plus lent partant des entrées et aboutissant à ce noeud.

On associe généralement à un circuit un graphe appelé graphe des délais où chaque sommet correspond à un composant (par exemple une porte logique), et où chaque arc correspond à une interconnexion entre les composants. A chaque sommet est associé un poids correspondant au délai de propagation du composant représenté.

Ainsi le chemin critique correspondra au chemin dont la somme des poids sera maximale. Ce dernier est appelé chemin critique topologique (topological critical path) car le comportement logique du circuit n'est pas considéré quand les chemins sont détermines. Le chemin critique topologique est distingué du chemin critique réel (true critical path) qui, lui considère le comportement logique du circuit. La différence réside dans le fait qu'un chemin critique topologique peut être un faux chemin (false path), c'est-à-dire un chemin où aucun signal ne peut se propager (voir figure 2.7). D'une façon générale le chemin critique réel possède un délai de propagation inférieur à celui du chemin critique topologique. En fait, le délai du chemin critique topologique est une borne supérieure de celui du chemin critique réel.

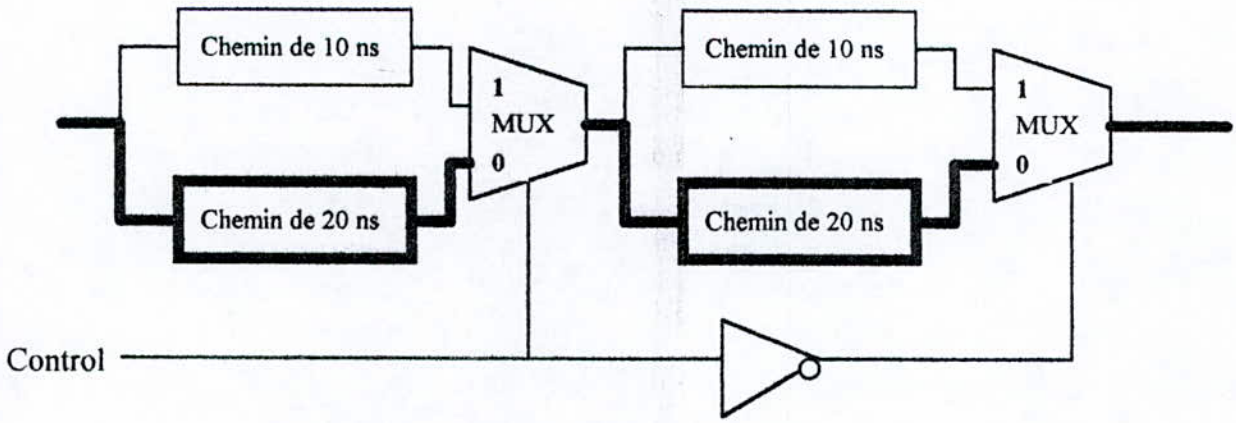


Figure 2.7 Exemple de faux chemin

Un chemin est appelé sensibilisable (sensitizable) s'il peut propager un signal partant de son origine jusqu'à son extrémité. Un chemin critique réel est un chemin sensibilisable de poids maximal. Soit un chemin $P = (V_{x1}, V_{x2}, \ldots V_{xn})$ où les x_i sont les sommets intermédiaires du chemin, la condition de sensibilisation du chemin P est la suivante [4]:

$$\frac{\partial f_{xi}}{\partial x_{i-1}} = (f_{xi} / x_{i-1} = 1) \oplus (f_{xi} / x_{i-1} = 0) = 1 \qquad \forall i = 1, n$$

ou $(f_{xi} / x_{i-1} = 1) = f_{xi}(x1, x2, \ldots, x_{i-2}, 1, x_i, \ldots, x_n)$

$(f_{xi} / x_{i-1} = 0) = f_{xi}(x1, x2, \ldots, x_{i-2}, 0, x_i, \ldots, x_n)$

Ce qui revient à dire qu'il existe au moins un vecteur aux entrées du circuit tel que le signal se propage jusqu'à l'extrémité du chemin. Dans ce cas on parle de condition de sensibilisation statique. Si de plus on considère que cette condition doit être vraie à chaque instant durant la propagation du signal, on parle de critère de sensibilisation dynamique. La condition de sensibilisation statique peut être considérée comme étant égale à la condition de sensibilisation dynamique quand le temps tend vers l'infini. En conséquence, le critère de sensibilisation statique peut conduire à une sous estimation du délai de propagation car certains chemins peuvent ne pas être statiquement sensibilisables mais peuvent quand même propager des signaux. Par conséquent, on peut considérer ce critère comme pouvant générer une borne inférieure du délai de propagation. Dans le cadre de l'optimisation de circuits, la condition de sensibilisation dynamique n'est pas utilisée, car l'approche indépendante de données fait que très peu d'informations sont connues concernant les signaux. Par conséquent des critères approchés tel que le critère de co-sensibilisation [22]sont plutôt utilisés.

11 Classification des méthodes d'optimisation

L'optimisation de circuits est en réalité intimement liée au processus de conception. A chaque étape, chaque niveau de description des problèmes d'optimisation sont posés aux concepteurs.

Parmi les méthodes d'optimisation de circuits les plus connues on peut citer: l'ordonnancement, l'allocation, la minimisation logique, le mapping, le placement & routage, la resynchronisation, le dimensionnement, la bufferisation, le compactage de layout, etc. Il est à noter que chacun de ces problèmes est en soi un domaine de recherche actif, pour améliorer toujours plus les performances des algorithmes et la qualité des circuits produits. Jusqu'à présent ces méthodes travaillent isolément et ne communiquent pas d'informations. Le but ultime sera de proposer une méthodologie unifiée qui intègre tous les niveaux de conception. Toutefois, il est clair que ce but ne sera pas atteint avant plusieurs années.

Etant donné la diversité des circuits existants et les différentes façons de les représenter, il n'existe pas de consensus quant à la classification des méthodes d'optimisation de circuits. Cependant, les auteurs s'accordent à faire la classification selon le niveau de représentation utilisé. On distingue généralement les niveaux suivants [5]:

le niveau structurel: l'optimisation consiste à modifier la structure du circuit pour atteindre les objectifs désirés. On peut par exemple utiliser la factorisation , la minimisation etc. Cette optimisation est parfois appelée resynthèse.

Le niveau topologique: l'optimisation consiste à modifier la position des composants au niveau du layout dans le but de réduire les temps de propagation de façon à ce que les composants critiques soient situés le plus prés possible les uns des autres.

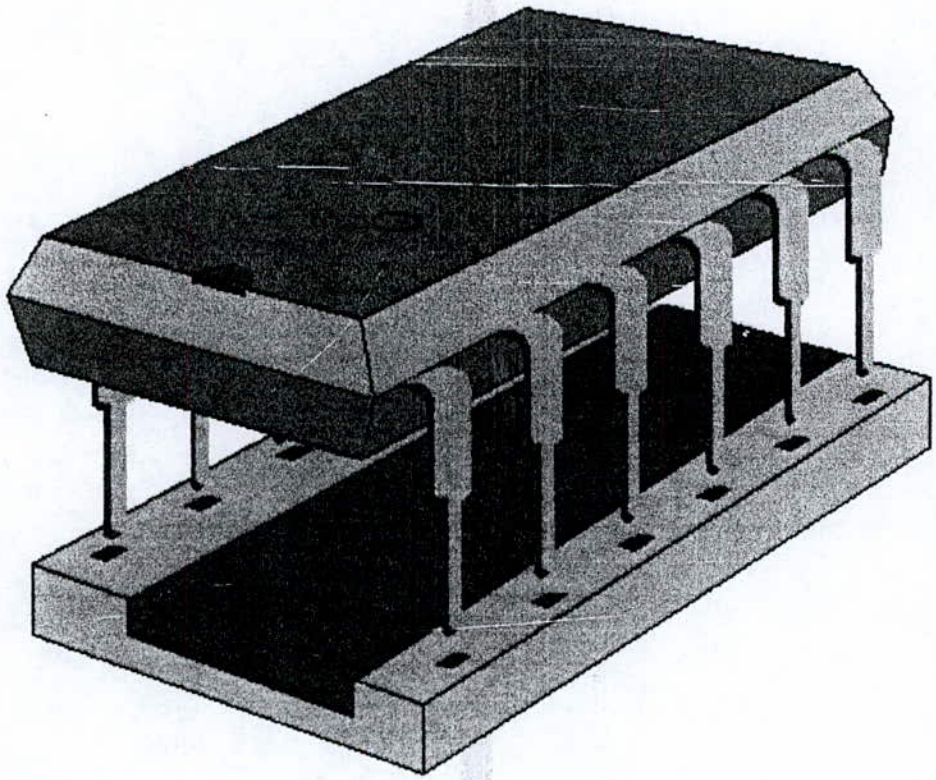
Le niveau physique: l'optimisation consiste à modifier les caractéristiques des composants des circuits, pour moduler les courants à travers les circuits afin d'atteindre les performances désirées, sans modifier la structure du circuit. Cette phase est généralement appelée "sizing" ou dimensionnement.

Le niveau physique qui est le thème de notre étude, se subdivise en deux catégories de problèmes [18]:

- **Le problème de dimensionnement discret** (ou dimensionnement de portes logiques) qui est, quant à lui, utilisé avec la méthodologie standard-cells. Il consiste à choisir pour chaque porte du circuit, dans une bibliothèque de cellules, parmi un nombre de cellules ayant des performances croissantes et des fonctionnalités identiques, la cellule qui aura les performances appropriées.
- **Le problème de dimensionnement continu** (ou dimensionnement de transistors) est utilisé avec la méthodologie full-custom, et consiste à choisir pour chaque transistor les paramètres qui lui permettent d'atteindre les performances requises. Le dimensionnement de transistors constitue un raffinement supplémentaire par rapport au dimmensionnement de portes logiques.

En résumé, notre étude consiste à développer des outils informatiques d'aide à la conception de circuits intégrés qui permettent l'optimisation temporelle de circuits combinatoires, au niveau physique (optimisation continue et discrète) en considérant essentiellement les paramètres temps et surface.

CHAPITRE III



Optimisation des circuits intégrés au niveau portes.

1 Problématique.

L'optimisation des circuits VLSI au niveau physique avec la méthodologie standard-cells se fait principalement en "modifiant" les dimensions des portes logiques. Cette modification se fait à travers le choix dans une bibliothèque, d'une cellule de caractéristiques appropriées. Formellement, il s'agit de choisir, pour tout le circuit à optimiser, une liste de cellules qui minimisent la surface totale et réduisent le délai de propagation du chemin le plus lent (et par conséquent, de tous les chemins du circuit), à être inférieur ou égal au délai maximal spécifié par le concepteur. La différence avec le problème de dimensionnement de transistors réside dans le nombre fixe de possibilités existant dans la bibliothèque. Il s'agit, par conséquent, essentiellement d'un problème de choix combinatoire.

2 Rappels sur la complexité d'algorithmes

La complexité d'algorithmes est une mesure de performance ou du temps d'exécution . La performance d'algorithmes est mesurée en fonction de la taille des données par une fonction notée T(n) . Etant donné que des algorithmes complexes peuvent traiter différents cas de figure, la performance est généralement estimée dans le cas le plus défavorable (worst case). On s'intéresse plus particulièrement au comportement asymptotique de la fonction T(n) pour avoir une idée des performances de l'algorithme quand la taille des données évolue. Pour cela, la notation 0 (grand O) est utilisée.

On dit que T(n) est en O(f(n)), s'il existe deux constantes c et no telles que :

$$T(n) \leq c\, f(n) \quad \forall\, n \geq no$$

Cette mesure permet aussi la comparaison d'algorithmes et la détermination approximative des tailles maximales des problèmes qui peuvent être traités par les algorithmes.

On considère généralement qu'un "bon" algorithme est un algorithme polynomial $0(n^k)$ parce que le temps de réponse de l'algorithme va évoluer de façon polynomiale avec la taille des données. Par contre, on considère un algorithme exponentiel comme étant un "mauvais" algorithme parce que le temps de calcul va vite devenir prohibitif, dès que la taille du problème augmente au delà d'un seuil.

2.1 Classification des problèmes.

On distingue les problèmes de décision pour lesquels le résultat donné par l'algorithme est oui ou non des problèmes d'optimisation dont la solution est mesurée par une fonction coût. Toutefois les problèmes d'optimisation peuvent être réduits à une séquence de problèmes de décision, et sont par conséquent aussi difficiles à résoudre que les problèmes décisionnels associés. Dans le cadre de la théorie de la complexité on s'intéresse particulièrement aux problèmes décisionnels, car ces derniers représentent une minoration de la complexité du problème associé.

La distinction entre algorithmes polynomiaux et exponentiels a permis d'établir une classification des problèmes. Si un problème possède un algorithme polynomial, alors on considère qu'il est traitable (tractable) ou encore qu'il appartient à la classe des problèmes P qui est considérée comme une classe de problèmes faciles. Si ce n'est pas le cas (i.e. si on ne connaît aucun algorithme polynomial), on considère généralement que ce problème est considéré comme un problème difficile.

Certains problèmes peuvent être résolus par des algorithmes polynomiaux qui sont exécutés sur des machines hypothétiques appelées machines Non-déterministes. Ces machines se caractérisent par un nombre infini de processeurs parallèles qui donnent la possibilité de faire systématiquement des choix optimaux en évaluant chaque possibilité à l'aide d'un processeur. Cette classe de problèmes est appelée classe NP (non-deterministic polynomial).

Une question posée depuis de nombreuses années par les spécialistes reste malheureusement ouverte est de savoir si P = NP. Il a toutefois été démontré qu'il existe des problèmes ayant pour propriété que si l'un d'eux pouvait être résolu à l'aide d'un algorithme polynomial, alors P=NP. Cette classe de problèmes s'appelle NP-difficile (NP-hard), tandis que la sous-classe de NP-difficile qui appartient à la classe NP s'appelle NP-complet (NP-complete). Ceci signifie que si un algorithme polynomial est trouvé pour un problème NP-complet ou NP-difficile, alors tous les autres problèmes de cette classe seront résolus en temps polynomial. Les spécialistes s'accordent maintenant à dire que cette éventualité est assez improbable.

Toutefois, pour certains problèmes NP-complets, on peut parfois exhiber un algorithme pseudo-polynomial qui a une complexité polynomiale dans des cas particuliers où la complexité est majorée par une fonction polynomiale de la taille du problème et de la valeur maximale des données. Ces problèmes pour lesquels il n'existe pas d'algorithmes pseudo-polynomiaux sont appelés fortement NP-difficiles (strongly NP-hard ou NP-hard in the strong sense) .

La démarche utilisée pour prouver qu'un problème appartient à la classe P consiste à exhiber un algorithme polynomial qui le résout. Alors que pour prouver qu'un problème appartient à la classe NP-complet, la démarche consiste à prouver que ce problème peut se réduire polynomialement à un problème appartenant à la classe NP-complet.

Dans le cas des problèmes intractables, on a généralement recours à des algorithmes approchés appelés heuristiques. Ces heuristiques vont essentiellement déterminer une solution qui n'est pas optimale (généralement, un optimum local) en un temps raisonnable (généralement avec une complexité polynomiale).

3 Modèles temporels

On distingue généralement les modèles suivants:

le modèle fixe et intervalle : Dans le premier , on représente le délai associé aux composants par un nombre (généralement un réel), tandis que dans le second le délai est représenté par une borne supérieure et une borne inférieure;

le modèle linéaire : Dans ce cas de figure, le délai de propagation est représenté par les paramètres servant au calcul du temps de propagation à l'aide d'une formule linéaire.

Le délai est calculé comme suit:

$$\textbf{délai} = R * Cout + \tau$$

où **R** **résistance équivalente de la porte**
Cout **capacité attaquée par la porte**
τ **délai intrinsèque de la porte**

Certains modèles considèrent pour chaque entrée vers chaque sortie une résistance différente. D'autres plus simples ne considèrent qu'une seule résistance associée à chaque porte.

Le choix du modèle temporel et de ses paramètres dépendra du degré de précision voulu et du temps de calcul toléré.

En fonction de la manière avec laquelle on considère l'effet de mémorisation des noeuds du circuit dû aux capacités parasites des circuits réels , on distingue :

Le mode transition (transition mode): dans lequel les noeuds conservent leurs valeurs jusqu'à ce qu'un nouvel état vienne remplacer l'ancien. Par conséquent, le calcul de délai nécessite deux vecteurs d'entrée, l'un pour l'état initial, l'autre pour le nouvel état ;

Le mode flottant (floating mode): dans lequel les noeuds sont considérés sans mémoire. Ils sont, ainsi, considérés comme ayant une valeur inconnue jusqu'à ce qu'un nouveau vecteur en entrée soit injecté.

Le choix du mode dépendra aussi du degré de précision voulu. En effet, le premier mode est le plus précis mais au prix de temps de calcul plus importants puisqu'il faudra considérer l'état initial, alors que dans le cas du mode flottant, les calculs seront plus simples et, donc, plus rapides à effectuer. Toutefois, le délai sera généralement surestimé.

Le délai dû aux interconnexions: Si on considère une interconnexion de longueur Lw et de largeur Ww, alors on lui associe une capacité Cw et une résistance Rw égales à:

$$Cw = (Lw * Ww) (\varepsilon / t)$$

$$Rw = Rs (Lw / Ww)$$

où ε **constante diélectrique**
t épaisseur de l'interconnexion
Rs résistance unitaire

Si on considère le délai entre deux composants dû à l'interconnexion qui les relie, alors celui ci est égal à (en utilisant le lumped-RC model):

$$\text{délai}_{\text{interconnexion}} = (R(\text{composant i}) + Rw) * (Cw + \sum_j \text{Cload(composant j)})$$

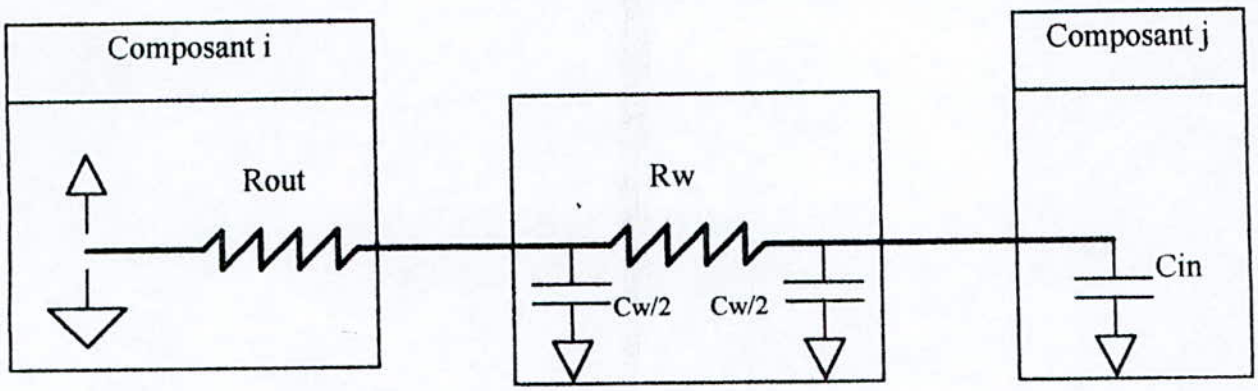


Figure 4.1 : illustration du délai des interconnexions

4 Formulation mathématique du problème de dimensionnement de portes

On considère que les circuits sont représentés par un graphe acyclique orienté G = (V,E). L'ensemble des sommets V est égal à {v1, v2, ..., vn}, tandis que l'ensemble des arcs E est inclus dans V*V. Chaque porte du circuit gi est représentée par un noeud vi appartenant à G. Chaque connexion du circuit reliant une sortie de la porte gi à une entrée de la porte gj est représentée par un arc du graphe (vi, vj). Les sommets ayant un degré d'entrée nul sont les entrées primaires du circuit. Alors que les sommets ayant un degré de sortie nul sont les sorties primaires. Par abus de langage, on entendra par chemin toute séquence d'arcs adjacents partant des entrées jusqu'aux sorties. On définit une fonction f qui associe chaque sommet vi au type de la porte gi associée. Le type d'une porte est représenté par un indice entier, référant au rang de la cellule dans la bibliothèque.

En résumé, un circuit combinatoire est représenté par un graphe G orienté acyclique et une fonction f. Une bibliothèque de cellules L peut être représentée par une famille d'ensembles finis L_1, L_2, ...L_k, où Li ($1 \leq i \leq k$) contient une ou plusieurs cellules logiquement équivalentes qui réalisent la porte de type i, k étant le nombre total de types de portes disponibles dans la bibliothèque.

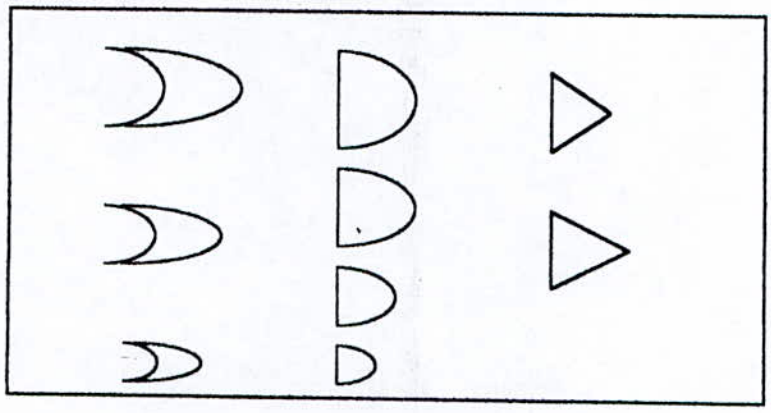


Figure 4.2 Illustration d'une bibliothèque de cellules

Chaque cellule sera caractérisée par sa surface et son délai. On associe, en outre à chaque cellule, pour chaque porte i :

$Li = [(a_{i1} , d_{i1}) , (a_{i2} , d_{i2}) ,.... (a_{ip} , d_{ip(i)})]$

qui est un ensemble fini de couples où $p_{(i)}$ représente le nombre de cellules réalisant la porte gi. Le symbole aij représente la surface de la j ème cellule réalisant la i ème porte. Le symbole dij représente le délai de la j ème cellule réalisant la i ème porte.

Etant donné un circuit décrit par G et f et une bibliothèque de cellules $L= \{ L_1, L_2, ...L_k \}$, une réalisation du circuit à l'aide de la bibliothèque est l'association de chaque sommet v appartenant à G avec une cellule de $L_{f(v)}$.

Soit S = (S1, S2,....Sn) une réalisation où $1 \leq Si \leq p_{f(v)}$. La surface du circuit à l'aide de la réalisation S est égale à : $\mathbf{A (S,G) = \sum_{i=1,n} a_{f(vi)} si}$

Soit un chemin vi1, vi2,....vil dans G. Le délai de ce chemin est égal à :

$$\mathbf{d(S,P) = \sum_{j=1,l} d_{f(vij)} si_j}$$

Le délai du circuit avec la réalisation S est noté:

D(S,G) = max { d(S,P) tel que P chemin de G }

Le problème de dimensionnement peut finalement être formulé comme suit:

Etant donné un graphe G, une fonction f, une bibliothèque L, et Tmax,

Minimiser **A(S,G)**

Tel que **D(S,G) ≤ T max**

5 Etat de l'art.

Les premiers travaux qui ont été entrepris dans le domaine de dimensionnement au niveau portes[20] [21] ont considéré des hypothèses simplificatrices sur les modèles de délai et sur la structure des circuits. Généralement, les délais de propagation des portes étaient considérés comme étant fixes et indépendants des charges.

Chan [20] considéra d'abord des cas particuliers de circuits ayant des topologies d'arbres et prouva un résultat théorique important, à savoir que, même dans ce cas de figure, le problème de dimensionnement est NP-difficile. Il proposa aussi un algorithme pseudo-polynomial pour les circuits ayant une topologie d'arbre. Puis il développa un algorithme qui transformait n'importe quel circuit en un circuit équivalent ayant une topologie d'arbre. Cette transformation s'opère par le biais de la duplication des portes (cloning). Malheureusement; cette méthode fonctionne uniquement avec des circuits de petite taille, car le nombre de portes ajoutées augmente exponentiellement.

Li [21] prouva un autre résultat théorique important qui consiste en l'affirmation suivante : Même lorsque les circuits sont restreints à des chaînes de portes, le problème de

dimensionnement est un problème NP-difficile. Li s'intéressa, lui aussi, à une classe particulière de circuits qui sont les circuits ayant une structure de graphe série/parallèle. Il proposa, à cet effet, un algorithme pseudo-polynomial. Par la suite, il [18] prouva que le problème de dimensionnement de portes logiques, quelle que soit la topologie du circuit, est NP-difficile au sens fort (NP-hard in the strong sense). Ce qui signifie qu'il n'existe pas d'algorithme pseudo-polynomial pour ce problème dans le cas général, à moins que NP=P. Cette démonstration est basée sur une réduction polynomial du problème 3-satisfabilité [9] au problème de dimensionnement.

A partir de ce stade, les travaux se sont tournés vers des méthodes approchées du problème et commencèrent, de plus en plus, à considérer le problème de dimensionnement dans sa généralité. Parmi les plus importants travaux, on peut citer :

- ceux de Moon [23] utilisant une heuristique qui transforme, à l'aide d'une méthode de cloning un circuit général en circuit ayant une topologie serie/parallèle. Ceci permet d'utiliser une méthode proche de celle de Li[18]. La différence qui existe est due au fait que puisque les modèles de délais utilisés considèrent les capacités de charge, une approche partant des sorties et allant vers les entrées est utilisée. Contrairement à l'algorithme de Li qui part des entrées vers les sorties. Ensuite la solution obtenue est affinée à l'aide d'une méthode de clonage (duplication des noeuds) plus efficace en termes de temps et d'espace. L'optimisation proprement dite est effectuée itérativement sur les chemins critiques. Ces travaux se caractérisent par l'utilisation d'un modèle de délai réaliste qui considère les charges.
- ceux de Fang [24] qui sont parmi les rares travaux qui considèrent la notion de vrais chemins critiques et qui n'optimisent pas uniquement localement sur chaque chemin critique. Bien au contraire, ils essaient de prendre en compte, en même temps, tous les

chemins critiques. Ceci est fait grâce à la notion de "delay contribution" qui est une mesure qui permet de choisir la porte dont l'optimisation aura le plus d'effet sur la performance du circuit. La "delay contribution" des portes est calculée uniquement sur les chemins critiques topologiques. Ensuite, quand une porte est sélectionnée, un test est fait pour savoir si elle appartient à un vrai chemin critique grâce à un algorithme appelé τ-Podem. Les chemins critiques sont calculés en utilisant l'approche "block oriented" [25] basée sur l'algorithme de PERT [26] qui permet de déterminer les chemins ayant une flexibilité (slack-time) nulle.

Toutefois, les travaux de Moon souffrent de l'imprécision des résultats, tandis que ceux de Fang se caractérisent par des temps de calcul élevés.. Il devient, alors, évident qu'un compromis devra être trouvé de façon à concilier la qualité des circuits optimisés et les temps nécessaires pour les calculs. Il est fort probable que là sera l'enjeu des futurs travaux de recherche en ce domaine.

6 Notre contribution dans le domaine du dimensionnement binaire (circuits mixtes CMOS-BiCMOS)

Dans le cadre de nos recherches, nous avons abordé un problème de dimensionnement binaire où le choix se fait entre deux technologies : CMOS et BiCMOS. Dans les paragraphes qui suivent, nous résumons les résultats essentiels de ces travaux, plus de détails sont présentés dans le chapitre 5 ainsi que dans l'article qui a été publié et reproduit en annexe 3.

Rappelons qu'un circuit BiCMOS est un circuit intégré qui contient à la fois des transistors MOS et des transistors bipolaires. Le but de notre travail est d'améliorer la performance d'un circuit en faisant pour chaque porte un choix entre une cellule associée CMOS ou BiCMOS.

Le choix devra permettre d'exploiter efficacement les qualités respectives des technologies CMOS et BiCMOS. Ces qualités sont :

- Pour la technologie BiCMOS , la vitesse élevée et la possibilité d'attaquer de grandes charges;
- Pour la technologie CMOS, la densité et la faible consommation d'énergie.

L'intérêt de rajouter à la technologie CMOS conventionnelle une technologie plus sophistiquée comme la technologie BiCMOS est que cette dernière offre des délais de propagation plus rapide. Toutefois, cet avantage n'est effectif que si la charge attaquée est supèrieure à un seuil appelé Cx. Ceci implique que si la charge attaquée par une porte est supérieure à Cx, il est préférable d'opter pour une cellule BiCMOS, car elle sera plus rapide. Par contre, si la charge est inférieure à Cx , il est préférable d'opter pour une cellule CMOS qui sera, alors plus rapide, tout en occupant une surface moins importante et consommera moins d'énergie

En effet, les cellules BiCMOS consomment typiquement 20% de plus que les cellules CMOS avec toutefois, une vitesse de 1,5 à 2 fois plus grande alors que les cellules CMOS. Par contre les cellules CMOS occupent généralement une surface 3 à 5 fois moins importante que celle des cellules BiCMOS.

Dans le but de tirer le meilleur parti possible de ces deux technologies à partir d'une bibliothèque de cellules mixtes (CMOS et BiCMOS), nous avons proposé, pour le choix de la cellule appropriée pour une porte donnée, une approche basée sur les règles suivantes :

1 Initialement, toutes les portes sont considérées par défaut en technologie CMOS dans le but d'augmenter la densité du circuit ;

2 Dans une seconde phase, toutes les portes attaquant des sorties sont choisies BiCMOS, étant donné les charges importantes des sorties;

3 Dans une troisième phase, les autres portes sont choisies en fonction de la charge attaquée par ces dernières. Cette charge est calculée comme étant la somme de toutes les capacités d'entrées des portes connectées ainsi que de la capacité d'interconnexion. Une fois calculée , elle est comparée à la valeur de Cx pour pouvoir effectuer le choix.

La capacité de charge dépend des cellules attaquées, étant donné que les capacités d'entrée des cellules diffèrent selon leur type. Aussi, la modification du type d'une cellule (ie de CMOS en BiCMOS) a pour effet de modifier les charges des noeuds connectés aux entrées de cette cellule. Il sera nécessaire, par conséquent, durant le processus de sélection des cellules, de traiter les portes en allant des sorties vers les entrées. Ainsi, les décisions prises ne seront plus remises en question par la suite car toutes les cellules attaquées auront elles aussi définitivement été sélectionnées.

6.1 Optimisation locale.

Nous avons d'abord proposé une approche locale à ce problème (voir annexe 3) . En effet, la décision de choisir le type de cellule est prise uniquement en fonction de la charge de la porte qui est une information locale à la porte considérée. Pour cela , nous avons utilisé un algorithme basé sur l'exploration BFS en partant des sorties du circuit. Cet algorithme permet de traiter les portes du circuit successivement dans leur ordre topologique décroissant, niveau logique après niveau. Cette approche permet d'avoir d'excellentes performances puisque les portes du circuit ne sont traitées qu'une seule fois, étant donné la complexité linéaire de l'algorithme BFS. L'algorithme BFS originel a toutefois été étendu pour pouvoir traiter les circuits réels. A cet effet, nous avons considéré la possibilité pour une porte d'avoir plusieurs sorties ainsi que la possibilité d'avoir des circuits possédant des boucles qui sont, notamment , utilisées pour les circuits mémoires. Les boucles sont traitées par une première passe qui les

élimine tout en mettant à jour les capacités des noeuds concernés. Cette élimination est obtenue grâce à un algorithme basé sur l'exploration DFS. Une fois que toutes les portes attaquées par une porte sont identifiées, la capacité de charge est exactement connue. Ce qui rend possible sa sélection. Ce processus prend fin lorsque les entrées du circuit sont atteintes.

Cette approche se caractérise par une complexité linéaire de l'algorithme (BFS et DFS) et, par voie de conséquence, sa grande vitesse qui lui permet de traiter des circuits de tailles importantes.

Au vu des résultats obtenus et qui sont reproduits dans la table1 du chapitre 5, les améliorations en terme de vitesse sont importantes (entre 10 et 40 %) et il a été noté que la surface additionnelle était généralement du même ordre de grandeur. L'examen d'un cas particulier où le gain en vitesse est de 40 % alors que la surface additionnelle n'est que de 6.2 % a permis d'entrevoir la suite des travaux. En effet, ces performances ont été obtenues car la sélection des cellules BiCMOS a été effectuée uniquement sur les chemins critiques. Ce qui a permis de diminuer sensiblement la surface additionnelle.

6.2 Optimisation globale.

Dans ce cas, on se propose de ne sélectionner des cellules BiCMOS que si la porte est située sur un chemin critique. Par conséquent, la sélection est faite en fonction des caractéristiques de tout le circuit . D'où l'appellation d'optimisation globale.

Le principe général demeure le même : La sélection se fait toujours en fonction de la capacité de charge et, par conséquent, se fait en partant des sorties vers les entrées, en utilisant les

mêmes règles. Une différence va être la possibilité pour le concepteur de spécifier le délai désiré du circuit et, ainsi, pouvoir considérer une contrainte sur le temps.

La méthode que nous avons proposée est itérative. D'abord, le chemin critique est déterminé ; ensuite, il est parcouru de sa sortie à son entrée, donc en sens inverse du sens de propagation des données. Pour chaque noeud rencontré, la comparaison est effectuée avec la valeur du Cx et la sélection de la cellule est faite. Une fois le noeud d'entrée rencontré, on sélectionne le nouveau chemin critique et le processus reprend. Il prend fin quand :

- il n'y a plus d'amélioration possible , c'est à dire lorsqu'on arrive à un chemin critique où aucune porte BiCMOS n'est sélectionnée,

- ou bien, quand la spécification du délai faite par le concepteur est atteinte, c'est à dire que le délai du chemin critique courant est inférieur ou égal à la spécification.

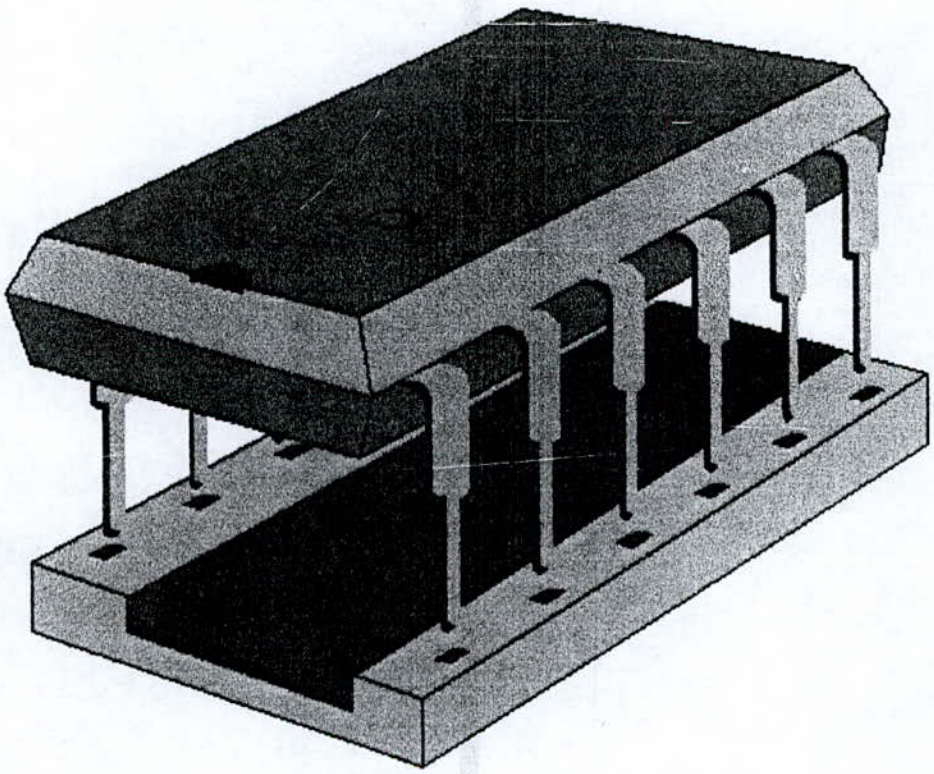
6.3 Conclusion

Nous avons pu mettre en oeuvre deux heuristiques pour le problème de dimensionnement: L'une locale effectuant une optimisation sans contraintes, l'autre globale effectuant une optimisation avec contraintes. Dans tous les cas l'approche que nous avons adoptée est une approche partant des sorties vers les entrées, une approche qui est aujourd'hui de plus en plus utilisée et que nous avons été parmi les premiers à adopter.

D'un point de vue théorique la complexité du temps de calcul de ces deux heuristiques est polynomiale, ce qui leur permet de traiter des circuits VLSI réels qui sont généralement de grande taille en fournissant des solutions acceptables. Toutefois les solutions obtenues sont

sub-optimales, étant donné que le problème de dimmensionement appartient à la classe des problèmes NP-Complets.

CHAPITRE IV



Optimisation de circuits intégrés au niveau transistor

1 Problématique

L'optimisation au niveau transistor est un raffinement par rapport à l'optimisation au niveau portes, puisque l'on descend au niveau de détail élémentaire qui est celui du transistor. L'optimisation de circuits VLSI au niveau physique avec la méthodologie full-custom se fait principalement en modifiant la taille des transistors. En effet, en agrandissant les dimensions d'un transistor, on augmente en même temps la quantité de courant pouvant passer à son travers et par conséquent la vitesse de propagation. Toutefois, à chaque fois qu'on agrandit les dimensions d'un transistor, il est clair qu'on augmente la surface du circuit ainsi que l'énergie consommée. D'ou la nécessité d'augmenter les dimensions des transistors avec modération pour minimiser la surface et la consommation d'énergie supplémentaire qui en résulterait. Pour cela, ne seront modifiées que les dimensions des transistors qui pourront améliorer significativement les performances. Ces transistors seront en l'occurrence, ceux situés sur les chemins critiques.

2 Rappel transistor MOS

Les transistors MOS (Metal Oxyde Semiconductor) sont à la base de la grande majorité des circuits intégrés fabriqués dans le monde entier, car leurs propriétés telles que la consommation statique très faible, leur régularité leur permettent de constituer des circuits de très grand niveau d'intégration. Ils sont formés par la présence sur un substrat de silicium de zones différemment dopées et surmontées d'une couche conductrice formant la grille. En

fonction du dopage, on distingue deux types de transistors MOS: Les transistors N et les transistors P. Les types N et P réfèrent à la polarité des porteurs dans le canal, qui est la même que celle de ceux du drain et de la source. Un transistor est caractérisé par trois grandeurs qui déterminent ses caractéristiques électriques. Ce sont la longueur et la largeur du canal (L et W) et l'épaisseur de l'oxyde de grille (D). Les transistors MOS sont dits à effet de champ, ce qui signifie que le champ électrique induit par la présence d'une tension sur la grille permet de moduler la résistance du canal. Par exemple, pour un transistor de type N dont les porteurs sont des électrons, il faut une tension de grille positive pour attirer les électrons de façon à former le canal.

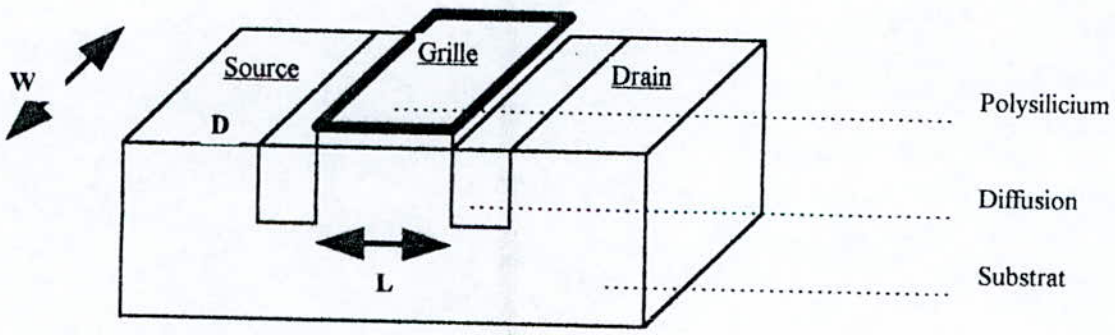


Figure 3.1 Schéma d'un transistor MOS

3 Modélisation des transistors

3.1 Modélisation classique

La quantité de courant Ids qui passe à travers le canal d'un transistor dépend des différentes tensions des noeuds de ce transistor. Ces tensions délimitent trois régions de fonctionnement décrites dans les équations de courant.

$$Ids = \begin{cases} 0 & Vgs - Vth < 0 \quad \text{(non passant)} \\ (b/2) * (Vgs-Vth)^2 & 0 <= Vgs - Vth <= Vds \quad \text{(saturé)} \\ b\,(Vgs - Vth)\,Vds - Vds^2 & Vgs - Vth > Vds \quad \text{(linéaire)} \end{cases}$$

où **Vgs** tension entre grille et source
Vds tension entre drain et source
Vth tension de seuil
b gain du transistor

Equations de transistor du premier ordre

Ces modèles se caractérisent d'abord par leur précision puis par la quantité importante de temps de calcul nécessaire pour en déduire par exemple des délais de propagation. En effet, ces délais sont calculés à partir de la résolution de systèmes d'équations différentielles déduits des modèles de transistors. Ce qui nécessite des temps de calcul importants particulièrement quand le système est de grande dimension. Cet état de fait a fait ressentir le besoin de modèles approchés de transistors. Dans le cadre de l'optimisation de circuits, les modèles approchés sont utilisés car [8]:

L'analyse étant indépendante de données en entrée (pattern indépendant analysis), très peu d'informations sont connues quant à l'évolution exacte des signaux d'entrée. Aussi l'utilisation de modèles temporels sophistiqués et donc coûteux en temps de calcul, n'est-elle pas justifiée. En outre la détermination de caractéristiques de circuit utilisées durant le processus d'optimisation telles que le chemin critique ne nécessite pas une précision absolue des délais de propagation, mais plutôt une précision relative de chaque bloc logique par rapport aux autres.

3.2 Modèle linéaire RC

Parmi les alternatives qui existent aux modèles basés sur les équations de transistors, on distingue le modèle RC linéaire qui est un des plus performants [30]. Ce modèle a été développé pour pouvoir simuler le comportement des transistors en régime transitoire, permettant ainsi de déterminer les temps de transition. Chaque transistor est modélisé à l'aide d'un interrupteur en série avec une résistance. Tandis que les noeuds de circuit sont représentés par des capacités.

3.2.1 Les capacités

Dans le but de prendre en considération l'effet capacitif des fils, on associe à chaque noeud la somme des capacités qui lui sont liées. Ces capacités sont en l'occurrence les capacités d'interconnexion des fils ainsi que les capacités parasites introduites par les transistors notamment les capacités de grille (Cgb, Cgd, Cgs respectivement capacités grille-substrat, capacités grille-drain, capacités grille-source), dont la somme est donnée par la formule suivante:

$$\mathbf{Cgrille = Cgb + Cgd + Cgs = Cox * (W*L)}$$

$$\mathbf{Cox = \varepsilon / Tox \quad (\varepsilon \text{ constante diélectrique, Tox épaisseur de l'oxyde})}$$

On considère aussi les capacités grille-drain et grille-source qui sont dues au phénomène de diffusion latérale. Cette capacité est formée par la contribution de la capacité de jonction de surface et de la capacité de jonction de périmètre. Ainsi, pour le drain et la source de chaque transistor, la capacité est approchée par la formule suivante:

Crec = Cja * (Overlap*W) + Cjp * 2 (Overlap+W)

Cja : capacité unitaire de jonction de surface.
Cjp : capacité unitaire de jonction de périmètre.
Overlap : valeur de recouvrement.

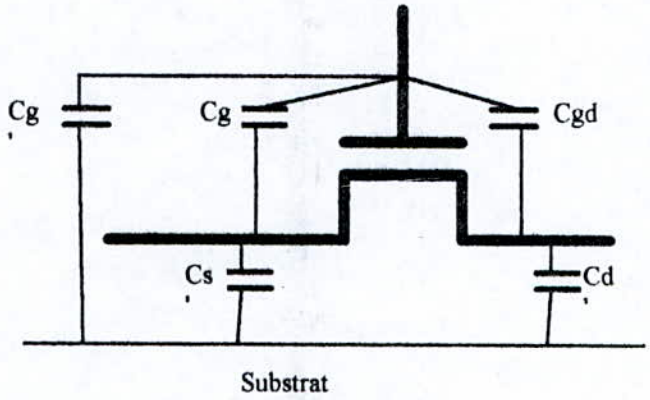


Figure 3.2 Capacités associées à un transistor MOS

3.2.2 Les résistances:

Les modèles les plus simples comportent une seule résistance tandis que les plus sophistiqués en comportent au moins deux en l'occurrence Rhigh et Rlow [11]. Dans ce dernier cas, elles sont utilisées suivant la nature de la transition qui peut être montante (rising transition) et dans ce cas c'est Rhigh qui est utilisée, ou bien descendante (falling transition) et dans ce cas c'est Rlow qui est utilisée. Cette modélisation permet de prendre en compte la nature asymétrique des transistors N et P qui peuvent en étant passant participer soit à:

- la charge de capacités et dans ce cas c'est le temps de montée qui est pris en compte,

la décharge de capacités et dans ce cas c'est le temps de descente qui est considéré.

Rdyn =
- **∞ si interrupteur ouvert**
- **Rhigh si interrupteur fermé et 1 logique propagé.**
- **Rlow si interrupteur fermé et 0 logique propagé.**

Ces résistances sont calculées à l'aide de tables de résistances unitaires qui permettent de calculer la résistance d'un transistor en fonction de son type et de ses dimensions ainsi que de la valeur logique du signal propagé. Une résistance est calculée selon la formule suivante:

$$R = Runitaire * (L/W)$$

Runitaire est une résistance unitaire par surface
L : Longueur du transistor
W : Largeur du transistor

4 Modèle de délai RC

Les transistors MOS sont les constituants élémentaires des "portes" logiques auquel est associé un délai de propagation. Par conséquent, il n'existe pas de délai proprement dit associé à chaque transistor, mais c'est plutôt l'ensemble des transistors d'une "porte" qui vont déterminer ce délai. Néanmoins on peut associer un délai à des circuits élémentaires composés de chaînes ou d'arbres élémentaires.

Les modèles de type RC sont en général utilisés pour l'estimation et l'optimisation des performances de circuits intégrés MOS. Il en existe une gamme variée, allant des plus simples comme le τ-model de mead-Conway [10], jusqu'à :

le lumped RC-model [11] où le délai est calculé comme suit sur la séquence de transistors la plus défavorable (worst case stage):

$$\text{délai} = \sum_i Ri * \sum_j Cj$$

(Ri résistance transistor i, Cj capacités du noeud j)

- le distributed Rc-model [11] où le délai est égal à:

$$\text{délai} = \sum_i R'_i * C_i$$

(R'_i résistance chemin de la source au noeud i)

- le Elmore RC-model amélioré[12] où le délai est calculé comme suit sur l'arbre de résistances le plus défavorable (Tw = worst case tree):

$$\text{délai} = \sum_{j \in TW} R_{oj} C_j + \tau /6 * (1 + 2\, V_{seuil}/ V_{DD})$$

(R_{oj} = Résistance chemin source-noeud O $\cap$ Résistance chemin source-noeud j)
(τ = temps de transition du signal appliqué aux entrées)

Les performances de ces derniers sont évidemment inversement proportionnelles à la précision obtenue. Par exemple, le dernier modèle est très précis puisqu' il prend en considération même la forme des signaux, au détriment du temps de calcul qui ,lui ,est plus important.

5 Formulation mathématique du problème de dimensionnement de transistors MOS

Pour un circuit combinatoire ce problème peut généralement être formulé comme suit:

minimiser la surface

avec pour contraintes:

délai < Tspécifié
dimension des transistors > taille minimale fixée par la technologie

En considérant wi la dimension du ieme transistor, on peut affiner la formulation sous la forme de fonctions posynomiales [12] qui ont pour propriété d'être convexes.

$$\text{Min} \sum_{i=1,n} wi$$

$$\text{Delai}(w) = \sum_{j=1,n} \gamma_j \prod_{i=1,n} x_i^{\alpha ij} < \text{Tspecifié}$$
$$wi \geq w_{min}$$

où : $\gamma_j \geq 0$

$\alpha_{ij} \in \{-1, 0, 1\} \ \forall\ i = 1,n \quad \alpha_{ij} = -1$ si transistor $i \in$ chemin critique

6 Etat de l'art de l'optimisation au niveau transistor:

Diverses méthodes ont été utilisées pour l'optimisation. Historiquement une des premières méthodes utilisées [13] était de type itératif ou incrémental. A chaque itération, le chemin critique est identifié, puis les dimensions des transistors qui le constituent sont optimisées. L'itération est initialisée en partant du circuit avec les dimensions minimales de transistors. Puis à chaque itération, la dimension d'un transistor choisi sur le chemin critique est augmentée. Le processus prend fin quand les contraintes temporelles sont satisfaites.

Une amélioration de cette méthode fut ensuite proposée [14]. Celle-ci traite les circuits MOS en deux phases. La première utilise la méthode itérative décrite ci-dessus pour obtenir une solution approchée initiale. Ensuite dans la seconde phase, le problème est modélisé mathématiquement en considérant uniquement le sous-circuit déterminé par le chemin critique. Ce qui permet de réduire la taille du problème modélisé qui est alors résolu localement à l'aide de la méthode des "feasible directions".

Ensuite, une autre approche [15] basée sur la formulation du problème sous forme non-linéaire fut proposée et le problème résolu à l'aide de la méthode de Lagrange. En même temps, fut aussi proposée la résolution du problème de transistor "sizing" à l'aide de méta-heuristiques telles que la méthode du simulated-annealing [16]. Dans tous les cas de figure, les approches décrites précédemment posent des problèmes quant à la qualité des résultats obtenus. Ces problèmes sont dus essentiellement :

- à la qualité des modèles temporels utilisés,
- au fait que ces méthodes sont basées sur des séquences d'optimisation locales qui considèrent, à chaque fois, des parties seulement de tout le circuit.

Une solution à ces problèmes a été proposée dans [12] , à travers une optimisation globale de tout le circuit avec un modèle temporel réaliste. Toutefois, les temps de calcul sont très importants et, en conséquence, cette méthode ne peut être appliquée qu'à des circuits de petite taille (inférieure à 1000 transistors).

7 Problème du flux de signal (signal flow)

Outre les problèmes précédemment cités, il est à noter celui de la nature bidirectionnelle des transistors MOS qui pénalise en termes de calcul, les outils CAO au niveau transistor et plus particulièrement les outils d'optimisation. Dans certains cas, ce problème peut même conduire à des résultats erronés. Aussi, la détermination automatique du sens de passage du flux à travers les transistors d'un circuit CMOS est-elle l'objet de travaux depuis une dizaine d'années pour, sans cesse, améliorer le rapport qualité/coût en termes de temps de calcul.

Généralement le principal traitement qu'effectuent les outils CAO au niveau transistors sur les circuits CMOS consiste en des explorations des transistors qui constituent le circuit à traiter. Ce processus est effectué dans le but de déduire des informations telles que le délai, l'état logique, la contrôlabilité etc ... , ou bien pour déterminer des sous circuits qui ont des propriétés intéressantes tels que les classes, les étages, les groupes etc ... Cependant, à cause de la nature bidirectionnelle des transistors MOS (contrairement aux transistors bipolaires qui sont unidirectionnels), ce processus d'exploration consomme la majorité du temps de calcul et peut même dans certains cas

aboutir à des résultats erronés. Par contre si le signal flow d'un transistor est connu, alors l'exploration de ce transistor se fait dans un seul sens (source vers drain ou drain vers source) ce qui a pour conséquence de réduire sensiblement le temps d'exploration. L'autre intérêt de la détermination du signal flow pour les outils CAO au niveau interrupteur, principalement les outils qui utilisent la notion de chemins, est la réduction du nombre de faux chemins (false paths). La conséquence de la réduction du nombre de ces faux chemins est d'augmenter la qualité des résultats.

Plusieurs types d'outils CAO au niveau transistors utilisent ou essayent de déterminer les signal flows des transistors CMOS. On peut citer par exemple, des analyseurs temporels, des analyseurs de testabilité, des ATPG, des abstracteurs fonctionnels et même certains simulateurs. Pour cela un pré-traitement des circuits CMOS est nécessaire pour identifier leur signal flow au préalable.

8 Développement d'une méthode pour solutionner le problème du flux de signal

Généralement les algorithmes existants et décrits dans la littérature sont classées en deux grandes catégories [13]. La première catégorie, constituée des méthodes algorithmiques, a pour principal avantage la vitesse d'exécution. La deuxième catégorie, constituée des méthodes basées sur la notion de règles (rule based), a pour principal avantage la flexibilité nécessaire pour traiter la grande variété de styles et de techniques de conception CMOS.

La méthode que nous avons proposée, qui est développée en détail dans l'annexe 1, est mixte (voir annexe 1). En d'autres termes, elle est algorithmique tout en étant aussi basée sur des règles. Cette approche réduit les principaux inconvénients de chacune des deux catégories, à

savoir la rigidité des méthodes algorithmiques et la lenteur des méthodes basées sur des règles. Elle permet également de traiter une grande variété de circuits grâce à des règles non tributaires de la technologie choisie et qui considèrent des effets globaux de circuits. Cette combinaison a permis d'obtenir des résultats satisfaisants

La principale caractéristique de notre approche est qu'un algorithme unique traite automatiquement tous les styles de conception , sans restrictions statiques, dynamiques, avec ou sans transistor bidirectionnel et ce , sans aucune intervention de l'utilisateur durant l'analyse. L'analyse exhaustive des chemins est évitée grâce à une approche de type "trace sélective" pour l'identification des transistors. Cette approche est basée sur un mécanisme récursif de propagation d'informations à travers les chemins explorés.

Notre approche ne nécessite pas de structures de données additionnelles telles que le ST-Graph. Par contre, notre algorithme opère directement sur la structure donnée du circuit, ce qui , par conséquent; réduit l'espace mémoire utilisé et évite les manipulations dynamiques de la mémoire.

Les autres caractéristiques qui contribuent à améliorer les performances de notre algorithme sont :

- l'utilisation de règles pour l'identification du flux de signal qui sont des règles topologiques et indépendantes de la technologie. De plus, ces règles ne produisent pas de résultats erronés.
- l'utilisation d'un mécanisme dynamique de sélection de règles qui permet de réduire le nombre de règles qu'il faut tester durant chaque phase de l'analyse.
- la notion de règles globales qui prennent en considération des informations relatives aux chemins par opposition aux règles locales qui considèrent uniquement un transistor et son voisinage immédiat.

- l'utilisation du concept de la force des noeuds pour la détermination du sens du flux.
- le mécanisme dynamique de partitionnement de circuit en trois régions qui permet de réduire le nombre de règles tout en augmentant la vitesse de traitement.

8.1 Principe

Notre algorithme est basé sur la méthode d'exploration en profondeur. Deux passes récursives sont appliquées sur les circuits. La première appelée Forward-pass est initiée à partir des noeuds d'entrée et essaye d'atteindre les noeuds de sortie; la seconde appelée backward-pass effectue les même opérations que la première passe, à la différence qu'elle est initiée à partir des noeuds de sortie, et qu'elle explore le circuit tant que la force des noeuds courants est inférieure à celle des noeuds atteints. Durant ces explorations, si un cycle est détecté, alors l'algorithme effectue un retour arrière et explore d'autres chemins.

L'algorithme suppose que tous les noeuds d'entrée et de sortie de chaque groupe de transistors sont identifiés et stockés dans des listes. Il est à noter que cette étape d'initialisations est effectuée de manière dynamique (intégrée dans l'algorithme) et que sa complexité est linéaire.

Il faut signaler également que la forward-pass est subdivisée en deux phases distinctes : la première basée, sur une extension des lois de Kirchoff, traite typiquement les séquences linéaires de transistors. La seconde, pour sa part, traite les configurations parallèles de circuits.

8.2 Complexité

Notre algorithme possède une complexité linéaire puisque, même dans les cas les plus défavorables, les transistors sont traités un nombre déterminé de fois. Dans la majorité des cas, les transistors sont traversés une seule fois, soit du drain vers la source soit de la source

vers le drain. Toutefois, pour les circuits hautement parallèles, les transistors sont traversés deux fois, une fois du drain vers la source, une seconde fois de la source vers le drain. Dans certains cas, les transistors sont traversés jusqu'à trois fois, typiquement avec une règle où la décision d'identifier le flux de signal est retardée jusqu'à l'exploration complète du noeud courant. La linéarité est obtenue grâce à l'utilisation de marqueurs booléens associés avec à la fois chaque noeud et chaque transistor du circuit. Chaque fois qu'un noeud est complètement exploré ou qu'un transistor est identifié, ils sont marqués pour éviter de les traiter par la suite.

8.3 Justesse de l'algorithme

La diversité des circuits CMOS est pratiquement illimitée, et chaque jour de nouveaux styles et astuces sont inventées. Pour pouvoir affirmer qu'un algorithme ne produit pas d'erreurs, le test des principaux styles de conception est indispensable mais insuffisant. Il est nécessaire d'établir formellement la justesse de l'algorithme dans tous les cas de figures. La preuve formelle de l'algorithme que nous avons développé, est traitée en détails dans l'annexe 2.

La démarche que nous avons adopté pour ce faire consiste à:

D'abord prouver que les informations nécessaires à l'identification sont correctement propagées par toutes les phases de l'algorithme. Ceci est fait pour la Forward-phase 1, en démontrant par récurrence sur le nombre de transistors et en considérant différentes configurations (série de transistors etc.). Pour la Forward-phase 2 et la Backward-pass, comme ces informations sont propagées durant le backtracking, la récurrence est établie sur les chemins en considérant les différentes configurations possibles (chemins simples, reconvergents etc.).

Ensuite nous avons démontré pour chaque règle qu'elle ne peut pas produire d'identification erronée. Ceci est fait par un raisonnement par l'absurde, qui consiste à supposer qu'il y a une erreur, et à prouver qu'on aboutit à une contradiction dans tous les cas de figure. La dernière étape a consisté à démonter que les forces des noeuds elles aussi ne pouvaient pas produire d'erreur. Pour cela nous avons considéré pratiquement toutes les combinaisons possibles, après avoir éliminé les combinaisons susceptibles de produire des erreurs, nous avons démontré par l'absurde qu'une erreur aboutissait dans tous les cas de figures à une contradiction.

8.4 Vérification expérimentale de l'algorithme

Les tests que nous avons effectué, ont porté sur un large éventail de styles et techniques de conceptions (voir table 2 de résultats au chapitre 5) telles que: des circuits statiques et dynamiques, des circuits à base de logique complémentaire, à précharge, à base de portes de transmissions etc.....

On peut citer: plusieurs additionneurs, plusieurs barrel-shifters, un multiplexeur-demultiplexeur, des latches statiques et dynamiques etc... Nous avons aussi testé avec succès tous les circuits réputés dans la littérature comme étant difficiles. On peut notamment citer les matrices de transistors de passage et le "tiny-Xor" qui sont longtemps restés un problème pour les outils CAO au niveau interrupteur. Dans la grande majorité des tests, tous les transistors ont été correctement identifiés. Les exceptions sont les circuits qui contiennent des configurations de transistors non série/parallèle. Etant donné leur structure non-conventionelle, généralement trois transistors unidirectionnels sont identifiés comme étant bidirectionnels. Il est à noter que notre algorithme ne produit pour autant pas d'erreurs, en identifiant des transistors bidirectionnels comme étant unidirectionnels.

Une solution au problème de ces transistors consisterait à étendre le nombre de forces de noeuds, au prix toutefois d'une augmentation sensible du temps de calcul, et surtout de la perte de la linéarité de l'algorithme. Etant donné que les circuits contenant des configurations non série/parallèles de transistors, sont peu utilisés, nous n'avons pas adopté cette solution.

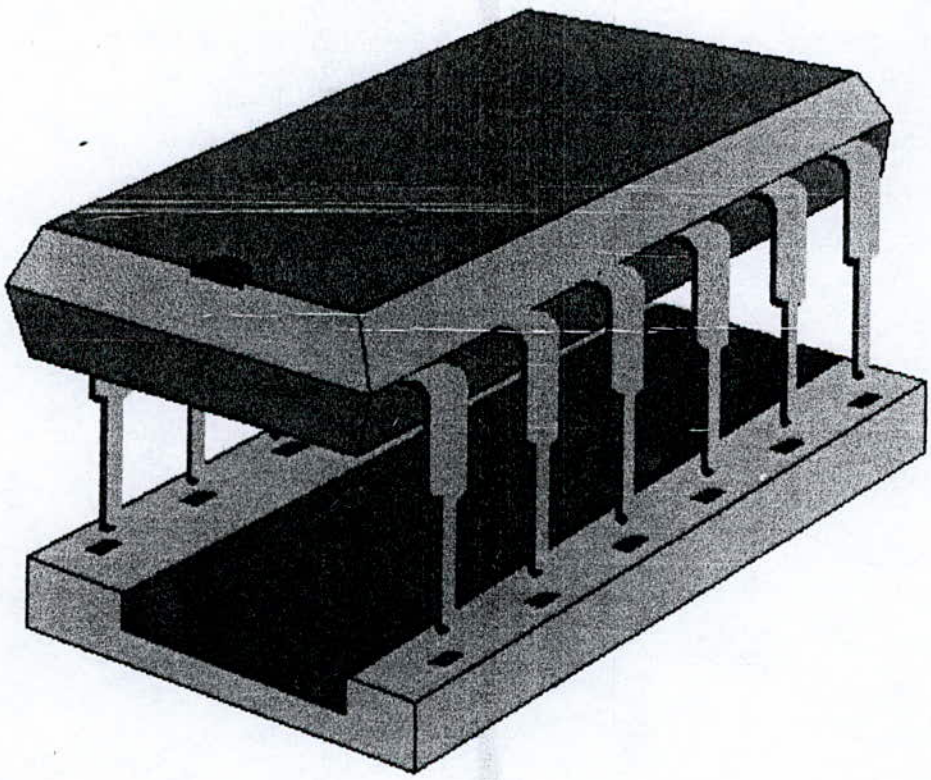
Les tests ont mis en évidence la grande vitesse de traitement de notre algorithme, en effet nous avons obtenu en moyenne une vitesse de l'ordre de 40000 transistors par seconde sur une station MIPS R3000. Les meilleurs temps ont été obtenus avec les circuits statiques, où la Forward-phase1 qui est la phase la plus rapide est appliquée à la majorité des transistors. Tandis que les temps les moins rapides ont été obtenus avec les circuits dynamiques et à base de portes de transmission où respectivement la Forward-phase2 et la Backward-pass qui sont les plus lentes ont été appliquées à la majorité des transistors.

La comparaison avec les autres approches décrites dans la littérature met en évidence des performances satisfaisantes (en moyenne 10 fois supérieures), qui sont dues principalement à la linéarité de notre algorithme.

8.5 Conclusion

Notre algorithme a été testé avec succès sur un échantillon représentatif des styles de conception de circuits VLSI full-custom. De plus, nous avons aussi testé avec succès un recueil de circuits reconnus dans la littérature comme étant difficiles ("pathological circuits"), les résultats sont présentés en détail dans le chapitre 5. De plus, nous avons pu contrairement aux autres méthodes prouver formellement la justesse de notre algorithme et, par voie de conséquence, assurer que notre algorithme ne produit pas de résultats erronés. Nous avons pu aussi prouver une propriété intéressante et unique de notre algorithme, à savoir qu'il possède une complexité linéaire du temps de calcul.

CHAPITRE V



Mise en oeuvre des méthodes développées :

Simulation et résultats

1 Simulation et résultats au niveau portes

1.1 Introduction

Comme il a été dit précédemment, il s'agit à ce niveau d'améliorer les performances temporelles de circuits standard-cells à travers le choix dans une bibliothèque, d'une cellule de caractéristiques appropriées CMOS ou BiCMOS.

La mise en œuvre des méthodologies proposées s'est faite sous l'environnement UNIX sur une station de travail MIPS R3000, à l'aide du langage C de programmation. Les tests ont été effectués sur les benchmarks ISCAS'85, qui sont des circuits standardisés pour le test des logiciels de CAO de synthèse et d'optimisation. L'estimation de la surface s'est faite grâce au logiciel de placement et routage Timber-Wolf. Tandis que l'estimation du délai maximum des circuits s'est faite grâce à un analyseur temporel que nous avons spécialement développé.

1.2 Règles et algorithmes utilisés

Rappelons qu'un circuit BiCMOS est un circuit intégré qui contient à la fois des transistors MOS et des transistors bipolaires. Dans le but de tirer le meilleur parti possible de ces deux

technologies en termes de performances et de surface de silicium, nous avons proposé, pour le choix de la cellule appropriée d'une porte donnée à partir d'une bibliothèque de cellules mixtes (CMOS et BiCMOS), l'approche suivante:

les portes sont choisies en fonction de la charge attaquée par ces dernières. Cette charge est calculée comme étant la somme de toutes les capacités d'entrées des portes connectées ainsi que de la capacité d'interconnexion. Une fois calculée en prenant en considération les dépendances entre les portes, elle est comparée à la valeur de Cx pour pouvoir effectuer le choix.

1.2.1 Première approche:

Nous avons au départ proposé une approche locale à ce problème. Le choix du type de cellule est fait en fonction de la charge de la porte (qui est une information locale à la porte considérée). L'algorithme que nous avons proposé permet de traiter les portes du circuit successivement dans leur ordre topologique décroissant, niveau logique après niveau en partant des sorties vers les entrées. La sélection d'une porte n'est effectuée qu'une fois que toutes les portes attaquées par cette dernière sont identifiées. Dans ce cas la capacité de charge de cette porte est alors précisément connue, ce qui rend possible sa sélection.

La méthode que nous avons proposée peut être schématisée comme suit:

- Mettre tous les noeuds de sortie dans la file d'attente
- TANT QUE cette file est non vide FAIRE
    - prendre le noeud de tête de la file
    - SI ce n'est pas une sortie ALORS
        - calculez sa capacité en fonction de celle de toutes les portes qui sont attaquées par ce noeud
        - identifier le type de la porte qui attaque ce noeud en conséquence
        - mettre à jour les capacités de ses noeuds d'entrée
        - mettre les noeuds d'entrée de cette porte dans la file d'attente
    - FSI
- FAIT

Figure 1 : Algorithme local de dimensionnement de circuits CMOS/BiCMOS

1.2.2 Deuxième approche:

Dans ce cas la sélection est faite en fonction des caractéristiques de tout le circuit. La sélection se fait plus précisément , exclusivement sur les chemins critiques.

Le principe de base est toujours le même : La sélection se fait en fonction de la capacité de charge et, par conséquent, se fait en partant des sorties vers les entrées, en utilisant les mêmes règles.

La méthode que nous avons proposée est itérative, et elle peut être schématisée comme suit:

1. D'abord, le chemin critique courant est déterminé.
2. Ensuite, il est parcouru de sa sortie à son entrée, donc en sens inverse du sens de propagation des données.
3. Pour chaque noeud rencontré, la comparaison est effectuée entre sa capacité avec la valeur du Cx et la sélection de la cellule est faite.
4. Une fois qu'un noeud d'entrée rencontré, on sélectionne le nouveau chemin critique et le processus reprend (reprise à partir de l'étape 2).
5. Le processus est arrêté quand:

- il n'y a plus d'amélioration possible , c'est à dire lorsqu'on arrive à un chemin critique où aucune porte BiCMOS n'est sélectionnée,
- ou bien, quand la spécification du délai faite par le concepteur est atteinte, c'est à dire que le délai du chemin critique courant est inférieur ou égal à la spécification.

1.3 Résultats obtenus

1.3.1 Résultats de la première approche

Au vu des résultats obtenus et qui sont reproduits dans la table 1, les améliorations en terme de vitesse sont importantes (entre 10 et 40 %) et il a été noté que la surface additionnelle était généralement du même ordre de grandeur. L'examen d'un cas particulier où le gain en vitesse est de 40 % alors que la surface additionnelle n'est que de 6.2 % a permis d'entrevoir la suite des travaux. En effet, ces performances ont été obtenues car la sélection des cellules BiCMOS a été effectuée uniquement sur les chemins critiques. Ce qui a permis de diminuer sensiblement la surface additionnelle.

Circuit	Nombre de portes	Nombre de portes BiCMOS	Amélioration de la vitesse	Surface additionnelle	Temps de calcul (ms)
C1355	620	58	11.2%	13.1%	19.1
C17	6	2	14.3%	35.8%	0.5
C1908	442	85	34.6%	15.7%	26.5
C2670	733	135	11.2%	11.4%	22
C432	180	20	40.9%	6.2%	5.7
C499	400	58	19.1%	18.8%	15.1
C880	263	57	12.3%	13.5%	11.9

Table 1

1.3.2 Résultats de la deuxième approche

La surface additionnelle obtenue avec l'approche globale est inférieure à celle obtenue avec l'approche locale (de l'ordre de 50% en moyenne). Ce résultat est obtenu grâce au plus petit nombre de cellules BiCMOS rajouté avec l'approche globale, étant donné qu'elles ne sont rajoutées que sur les chemins critiques. Par contre, le temps de calcul de l'approche globale est quand à lui beaucoup plus important (au moins 10 fois plus), étant donné que l'algorithme nécessite à chaque itération, la détermination du chemin critique.

1.4 Conclusion

La mise en œuvre des méthodes que nous avons développées pour l'aide à l'optimisation de circuits intégrés au niveau portes (méthodologie standard-cells), a permis de vérifier les performances théoriques des heuristiques.

La comparaison des résultats obtenus par les deux heuristiques a permis de montrer que la différence se situait essentiellement au niveau de la surface additionnelle et du temps de calcul. En effet la surface additionnelle obtenue avec l'approche globale est inférieure à celle obtenue avec l'approche locale (de l'ordre de 50% en moyenne). Ceci est du au fait que le nombre de cellules BiCMOS rajouté avec l'approche globale est plus petit étant donné qu'elles ne sont rajoutées que sur le chemin critique. Le temps de calcul de l'approche globale est quand à lui plus important (au moins 10 fois plus) étant donné la nature itérative de l'algorithme qui à chaque itération, nécessite la détermination du chemin critique.

Les résultats obtenus montrent que des améliorations de la vitesse des circuits allant de 11% jusqu'à 40% sont obtenues. Il est à noter en outre, que ces améliorations ont été obtenues avec un minimum de surface additionnelle, et ce avec des temps de traitement de l'ordre de quelques secondes. Cette dernière constatation permet d'affirmer que nos heuristiques peuvent être appliquées à des circuits VLSI réels.

2. Simulation et résultats au niveau transistor

2.1 Introduction

Rappelons que l'optimisation de circuits VLSI au niveau physique avec la méthodologie full-custom se fait en considérant les transistors. Nous avons développé, dans ce domaine, une méthodologie qui s'appuie sur la résolution d'un problème sous-jacent au problème d'optimisation, appelé problème de détermination des sens de flux des transistors. La

résolution de ce problème permet une détermination plus efficace et plus précise des chemins critiques nécessaires au processus d'optimisation.

La mise en œuvre des méthodologies proposées s'est faite sous l'environnement UNIX sur une machine MIPS R3000, à l'aide du langage C de programmation. Les tests ont été effectués sur un large éventail de benchmarks décrits en format SPICE, qui rassemble la plupart des circuits référencés dans la littérature qui ont été utilisés pour tester des algorithmes de détermination de signal flow.

2.2 Règles et algorithmes utilisés

Rappelons que les algorithmes existants et décrits dans la littérature sont classés en deux grandes catégories. La première catégorie, constituée des méthodes algorithmiques, a pour principal avantage la vitesse d'exécution. La deuxième catégorie, constituée des méthodes basées sur la notion de règles (rule based), a pour principal avantage la flexibilité nécessaire pour traiter la grande variété de styles et de techniques de conception CMOS.

La méthode que nous avons proposée, est mixte. En d'autres termes, elle est algorithmique tout en étant aussi basée sur des règles, ce qui permet de combiner les principaux avantages de chacune des deux catégories. Le principe général est de chercher et propager les informations disponibles sur les entrées et sorties, puis à les exploiter en tenant compte des propriétés des circuits CMOS pour identifier le flux de signal des transistors.

Concernant l'aspect règles de notre approche, celles que nous avons proposées sont les suivantes:

1. Si un transistor est directement connecté à un noeud d'entrée, alors son signal flow ne peut qu'être sortant du noeud d'entrée.
2. Si un seul transistor connecté à un noeud est non identifié, alors que tous les autres signal flows sont entrants, alors le flow de ce transistor est sortant.
3. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud d'entrée et atteignant un transistor non identifié (plus précisément un de ces noeuds diffusion), en plus si ce transistor est connecté à un noeud mixte ou noeud destination, alors son signal flow est un flow sortant.
4. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud d'entrée et atteignant un transistor non identifié (plus précisément un de ces noeuds diffusion), si tous les chemins arrivant à ce transistor n'atteignent que des noeuds sources, alors le flow de ce transistor est entrant.
5. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud d'entrée et atteignant un transistor non identifié (plus précisément un de ces noeuds diffusion), si tous les chemins arrivant à cc transistor n'atteignent que des noeuds mixtes ou de sortie, alors le flow de ce transistor est entrant.
6. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud d'entrée et atteignant un transistor non identifié, si tous les chemins arrivant à ce transistor n'atteignent que des noeuds mixtes ou d'entrée, alors le voisinage du noeud courant relativement au transistor non identifié est examiné. Si tous les chemins partant du noeud voisin atteignent des noeuds d'entrée ou de sortie, alors le transistor est bidirectionnel, autrement son flow est identifié comme étant entrant relativement au noeud courant.
7. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud de sortie et atteignant un transistor non identifié, si en plus ce transistor est connecté à un noeud

mixte ayant tous les transistors qui lui sont connectés identifiés, Alors le flow de ce transistor est sortant.

8. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud de sortie et atteignant un transistor non identifié, si tous les chemins arrivant à ce transistor n'atteignent que des noeuds de sortie, alors la même procédure utilisée pour la règle numéro 6 est appliquée.
9. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud de sortie et atteignant un transistor non identifié, si tous les chemins arrivant à ce transistor n'atteignent que des noeuds mixtes ou d'entrée, alors le flow de ce transistor est sortant.
10. Étant donné un chemin d'exploration (ou direction d'analyse) partant d'un noeud de sortie et atteignant un transistor non identifié, si tous les chemins arrivant à ce transistor n'atteignent que des noeuds mixtes ou d'entrée, si en plus la force du noeud courant est plus grande que celle du noeud précédemment exploré, alors le flow de ce transistor est entrant.

Notre algorithme opère en deux passes. Schématiquement la première part des entrées et essaye d'aller le plus profondément à l'intérieur du circuit. Tandis que la deuxième quant à elle, part des sorties et elle aussi essaye d'aller le plus profondément à l'intérieur du circuit. Dans les deux cas on détermine les forces des noeuds traités, on propage des informations sur les chemins explorés, pour pouvoir finalement identifier le signal-flow des transistors rencontrés.

Schématiquement l'algorithme opère comme suit :

- POUR chaque groupe de transistors du circuit
  - Effectuer la phase d'initialisation
  - Effectuer la première passe appelée forward-pass, (qui comprend deux phases)
  - Effectuer la deuxième passe appelée backward-pass
- FAIT

Figure 2 : Algorithme général de détermination du signal-flow

2.3 Résultats obtenus

Nous avons testé la plupart des circuits disponibles dans la littérature, y compris ceux réputés difficiles appelés communément les cas pathologiques. Les circuits ont été choisis de façon à être représentatifs des differents styles et techniques de conception des circuits CMOS. Dans tous les cas nous avons obtenus des résultats non erronés, et ce avec une rapidité qui permet d'appliquer notre approche à des circuits VLSI réels.

2.4 Conclusion

Les résultats obtenus comparés à ceux présentés dans la littérature sont satisfaisants puisque notre méthode est au moins 10 fois plus rapide tout en offrant une grande précision. Plus précisément notre algorithme permet de traiter en moyenne plus de 40000 transistors/seconde. La vitesse de traitement est encore plus grande pour les circuits statiques (50000 transistors/seconde) car les phases les plus rapides de l'algorithme sont appliquées à la majorité des transistors du circuit. Ces phases sont celles destinées à traiter les chaînes linéaires de transistors issues des noeuds sources qui sont beaucoup plus nombreuses pour les circuits statiques. Par contre pour les circuits dynamiques, les phases les plus lentes de l'algorithme sont appliquées à la majorité des transistors du circuit, ce qui par conséquent

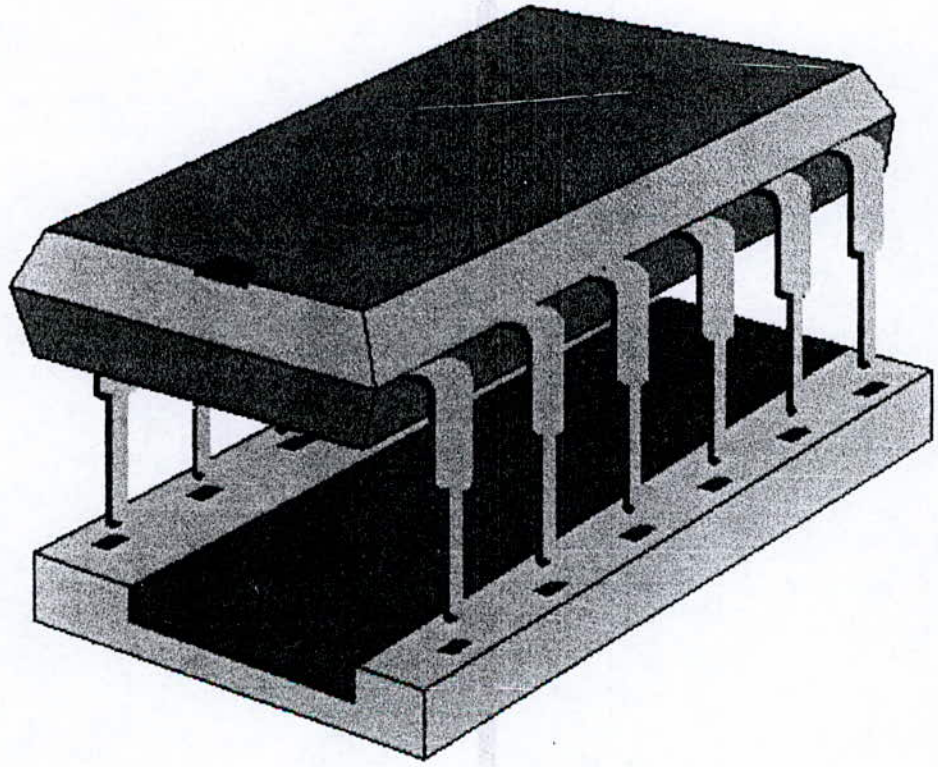
pénalise le temps de calcul (30000 transistors/seconde). Ces phases sont celles destinées à traiter les combinaisons parallèles de transistors. Celles-ci sont plus sollicitées étant donné la présence des transistors de précharge et d'évaluation supplémentaires des logiques dynamiques.

Circuit	Référence	Commentaire	nombre de transistors unidirectionels	nombre de transistors bidirectionels	nombre de transistors unidirect. Identifiés bidirect.	CPU time (ms)
1	[19] page 312	32 bits Complementary adder	1216	0	0	21.8
2	[19] page 323	Carry lookahead adder	18	0	0	0.5
3	[19] page 316	32 bits Dynamic adder	704	0	0	19.6
4	[19] page 324	Manchester carry chain	21	0	0	0.6
5	[19] page 173	Transmission gates MUX-DEMUX	16	0	0	0.4
6	[19] page 318	32 bits Transmission gate adder	704	0	0	23.9
7	[11] page 341	Barrel shifter	20	0	0	0.6
8	[11] page 342	Bidirectional Barrel shifter	24	16	0	1.2
9	[19] page 350	Static CMOS RAM cell	6	2	0	0.2
10	[19] page 354	Parity generator	18	0	3	0.6
11	[19] page 495	Circuit with bidirectional I/O ports	14	0	0	0.5
12	[19] page 216	Cross coupled fflop	16	0	0	0.5
13	[19] page 361	Column tree decoder	14	0	0	0.3
14	[19] page 364	Diffenretial Sense amplifier	14	0	0	0.5
15	[19] page 170	CVSL Xor	17	0	3	0.6

Table 2

CHAPITRE VI

CONCLUSIONS
&
PERSPECTIVES



Conclusion Générale et Perspectives.

Dans le cadre de notre étude, nous avons abordé le problème du dimensionnement au niveau physique sous ces deux aspects:

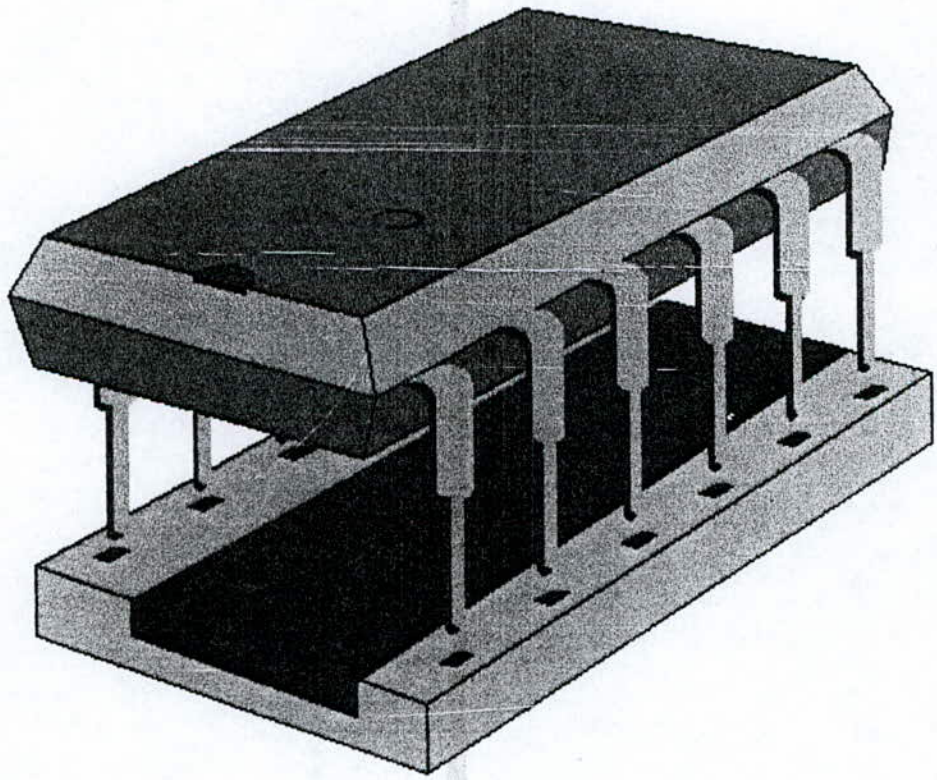
- Au niveau transistor, nous avons proposé une nouvelle approche pour déterminer le sens du flux de signal à travers les transistors et les résultats obtenus comparés à ceux présentés dans la littérature sont satisfaisants. De plus, nous avons pu établir formellement la justesse de notre algorithme .
- Au niveau porte, nous avons proposé deux heuristiques pour l'optimisation de circuits CMOS/BiCMOS. Nos heuristiques sont basées sur une approche (backward) partant des sorties jusqu'aux entrées qui est, aujourd'hui , de plus en plus répandue. En outre, contrairement à la plupart des approches qui opèrent sur des circuits simples, nous avons pu traiter les circuits réels possédant plusieurs sorties et des boucles,.

Le handicap de l'approche globale qui est le temps de calcul pourrait être réduit en minimisant le temps consommé à chaque itération, par la détermination du chemin critique. Pour cela, la possibilité de développer des algorithmes incrémentaux de chemins critiques devrait être sérieusement étudiée. De tels algorithmes pourraient à la condition de pouvoir considérer la notion de vrai chemin critique, sensiblement améliorer le rapport qualité circuit/temps de calcul qui est le principal problème rencontré par les optimiseurs de circuits au niveau physique.

En abordant le problème de l'optimisation au niveau physique, nous nous sommes concentrés sur la question du dimensionnement. Néanmoins toujours au niveau physique, il existe d'autres approches telles que celles de l'insertion de buffers, la minimisation de fanout, etc... Il est clair que si ces approches étaient intégrées dans un seul algorithme , ce dernier produirait des circuits encore plus performants. Malheureusement, comme l'a signalé Li [21] , il semble qu'il soit très difficile de considérer tous ces paramètres simultanément dans l'immédiat.

Il serait aussi, à long terme, intéressant de considérer le problème de l'optimisation non plus seulement au niveau physique mais également, à tous les niveaux à la fois, en partant des niveaux les plus élevés (synthèse de haut niveau) en considérant l'aspect performance pour l'ordonnancement [32] et l'allocation [2], au niveau structurel pour la synthèse [1], la resynchronisation [28] etc et enfin au niveau topologique. Nous suggérons une méthodologie hiérarchique pour l'optimisation de CI qui pour être efficace devra considérer même aux niveaux les plus élevés les caractéristiques physiques des circuits [29] [31]. Une telle méthodologie devrait pouvoir offrir des mécanismes de communication entre la hiérarchie d'algorithmes d'optimisation de façon à en permettre l'inter-complémentarité. Toutefois dans l'immédiat, il s'agit d'améliorer les performances des algorithmes pris isolement ce qui en soit, représente une tache délicate à laquelle nous espérons avoir apporté notre humble contribution.

BIBLIOGRAPHIE



Bibliographie

[1] G. De Micheli
Synthesis and optimisation of digital circuits, McGraw-Hill inc., 1994.

[2] D. Gajski et al
High level synthesis: Introduction to chip and system design, Kluwer-academic, 1992

[3] G. Kamarinos et al
How will physics be involved in silicon microelectronics, Journal of physics D, Applied Physics, pp 487-500, 1996

[4] M. Breuer
Diagnosis & reliable design of digital systems, computer science press inc., 1976.

[5] Hsi-Chuan Chen et al
Critical path selection for performance optimisation, IEEE trans. On CAD, Feb 1993

[6] C. Papadimitriou et al
Combinatorial Optimisation, algorithms and complexity, Prentice-hall 1996

[7] S. Even
Graph algorithms, computer science press, 1979

[8] Norman P. Jouppi
Derivation of signal flow direction in MOS VLSI, in IEEE Trans. on CAD vol. 6, No3, pp 480-490, May 87.

[9] Garey & Johnson
Computers and Intractabilité: a guide to the theory of NP-completeness, Freeman Company, 1979

[10] Mead & Conway
Introduction to VLSI systems, Addison Wesley, 1980.

[11] John K. Ousterhout
A switch-level timing verifier for digital MOS VLSI, in IEEE Trans. on CAD Vol CAD-4 No 3, July 85.

[12] S. Sapatnekar et al
An exact solution to the transistor sizing problem for CMOS circuits using convex optimisation, IEEE Trans. On CAD, November 1993.

[13] J.P Fishburn et al
Tilos: a posynomial programming approach to transistor sizing, in proc. ICCAD, pp 326-328, 1985.

[14] J. Shun et al
Optimisation-based transistor sizing, IEEE Journal of Solid-state circuits, pp 400-409, Apr. 1988.

[15] D. Marple
Transistor Size optimisation in the Taylor layout system, in 26 th DAC, pp 43-48, 1989.

[16] Chang et al
Prompt-3 : A cell based transistor sizing program using heuristic and simulated anealing, in CICC (custom integrated circuit conference) 1989.

[17] J. Monteiro et al
Estimation of average switching activity in combinational logic circuits using symbolic simulation, IEEE trans. On CAD, pp 121-127, Jan. 1997.

[18] Wing Ning Li
Strongly NP-Hard discrete Gate sizing problem, IEEE trans on CAD , August 1994.

[19] N.West, K.Eshraghian
Principles of CMOS VLSI design, a system perspective, Addison Wesley, 1985.

[20] Chan
Algorithms for library-specific sizing of combinational logic, 27th DAC , pp 353-356, 1990.

[21] W. Li et al
On the circuit implementation problem, IEEE trans on CAD, pp 1047-1051, Aug. 1993.

[22] S. Devadas et al
Computation of Floating mode delay: Theory and algorithms, IEEE trans on CAD, pp 1913-1923, Dec. 1993.

[23] Moon et al
A path-oriented algorithm for the cell selection problem, IEEE trans on CAD, pp296-307, March 1995.

[24] Chen-Ling Fang
Timing optimisation by gate resizing and critical path identification, IEEE trans on CAD, 1995.

[25] R.B. Hitchcock et al
Timing analysis of computer hardware, IBM Journal of research develop. pp 100-105, Jan.1982.

[26] T. Kirkpatrick et al
PERT as an aid to logic design, IBM Journal of research develop. pp 135-141, Mar.1966.

[27] W. Chuang
Timing and area optimisation for standard cell VLSI circuit design, IEEE trans on CAD, pp 308-320, March 1995.

[28] Leiserson et al
Retiming synchronous circuitry, Algorithmica VOL 6, No 3, pp 5-35, 1991.

[29] C. Ramachandran et al
Combined topological and functional-based delay estimation using a layout-driven approch for high-level application, IEEE trans on CAD , Dec.1994.

[30] R. Bryant
A survey of switch level algorithms, IEEE design & test, pp 26-40, Aug 1987.

[31] Tolga Soyota et al
Incorporating interconnect register and clock distribution delays into the retiming process, IEEE trans on CAD , Jan. 1997.

[32] Khelladi, Baba-Ali et al
A comparative study between the simulated anealing methods applied to the high level synthesis of integrated circuits, International conference on signal and systems, ICCS'94, Algiers,1994.

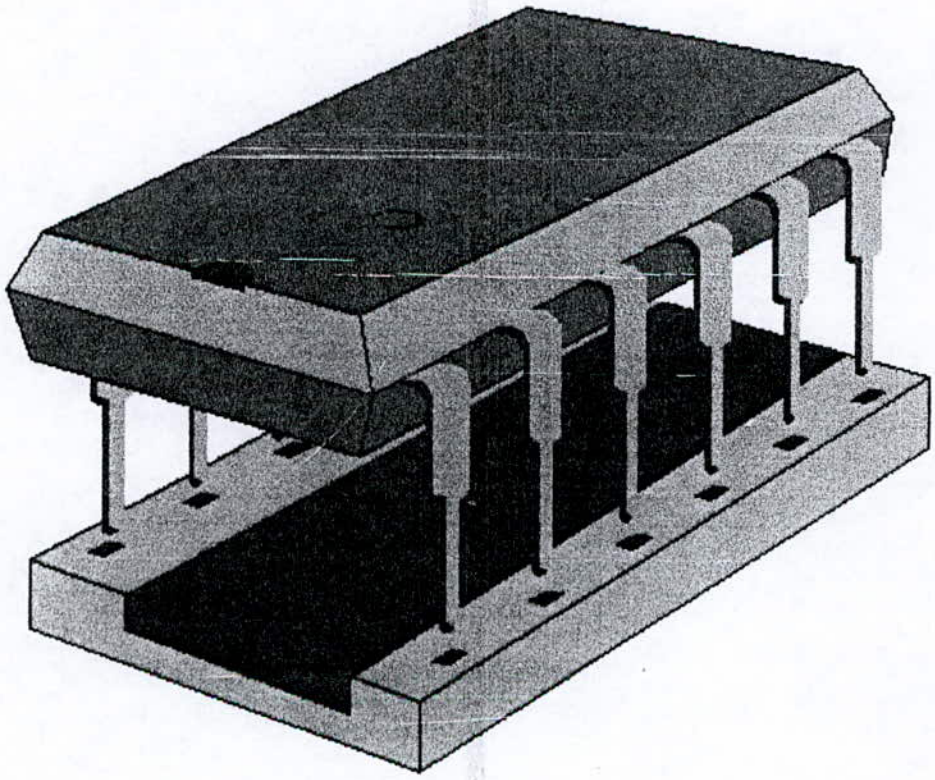
[33] A.R. Baba-ali, A. Farah
An efficient algorithm for signal flow determination in CMOS/VLSI, in proceedings IEEE EDTC'96 (European Design & test Conference), Paris 1996.

[34] A.R. Baba-ali, A. Farah
An efficient method for the derivation of signal flow direction in digital CMOS/VLSI, IEICE Transactions on fundamentals of electronics, communications and computer Sciences, Special section on VLSI design and CAD algorithms, Oct. 1997.

[35] A.R. Baba-ali, A. Bellaouar
A delay optimisation CAD tool for mixed CMOS/BiCMOS standard cells circuits, AJSE (Arabian Journal of Science & Engineering) Special issue on microelectronics, Oct. 1994

[36] A.R. Baba-ali, A. Farah
Formal proof of correctness of an efficient algorithm for signal flow determination in digital CMOS/VLSI, submited to IEEE Trans. On computer aided design of VLSI Circuis and systems

ANNEXES



Annexe 1

A Mixed Algorithmic and Rule-Based Approach For Signal Flow Determination In Digital CMOS VLSI

Abstract

Signal flow determination of CMOS/VLSI digital circuits is a key issue for switch-level CAD tools such as timing and testability analyzers, functional abstractors, ATPGs etc .. and even some simulators. Signal flow determination is used to preprocess circuit MOS transistors in order to improve both the accuracy and the running time of these CAD tools. Existing algorithms can be classified into two main categories: the rule-based approach and the algorithm-based approach. However, both of them have several drawbacks.

This paper presents an efficient algorithm based on a novel mixed algorithmic and rule based approach. Our algorithm overcomes most of the drawbacks of the pure algorithmic and rule based approaches. It is based on a set of «safe » topological rules rather than ad hoc or technology dependent ones, while the algorithmic aspect of our approach is based on a recursive Depth First Search (DFS). Due to the algorithmic aspect of our approach, some rules consider circuit global effects such as path informations. Our approach provides the advantages of the rule based one (ie : the flexibility and the adaptability toward the great variety of CMOS design styles) as well as the advantages of the algorithmic approach (ie: the fast processing time and the ability to consider circuits global effects). The result is that the software is very accurate since all the unidirectional and bidirectional transistors are correctly identified in all the pathological benchmarks reported in the literature. In addition, the software is fast (about 40 000 transistors/second) with a linear processing time.

Extended version of the paper published in IEICE Transactions on fundamentals of electronics, communications and computer sciences, special issue on VLSI design October 1997

1 Introduction

Generally MOS switch level CAD tools perform on circuits a sort of tree walk processes, where circuit transistors are explored and parsed many times. This process is performed in order to derive informations such as delay, logic state, controllability etc .., or to find subcircuits sharing common properties such as classes [1], groups[1] or stages [3].

The signal flow of a transistor is defined as the direction from signal source to signal destination [2]. Due to the bidirectional nature of MOS transistors, this process can consume a large amount of the total running time of the CAD tool, and can also lead to erroneous results [2,3] if transistor signal flows are not precisely known. If a transistor is known to be unidirectional then it is only necessary to explore either the source or drain, and hence this can substantially reduce the search time. Another benefit of identifying unidirectional transistors

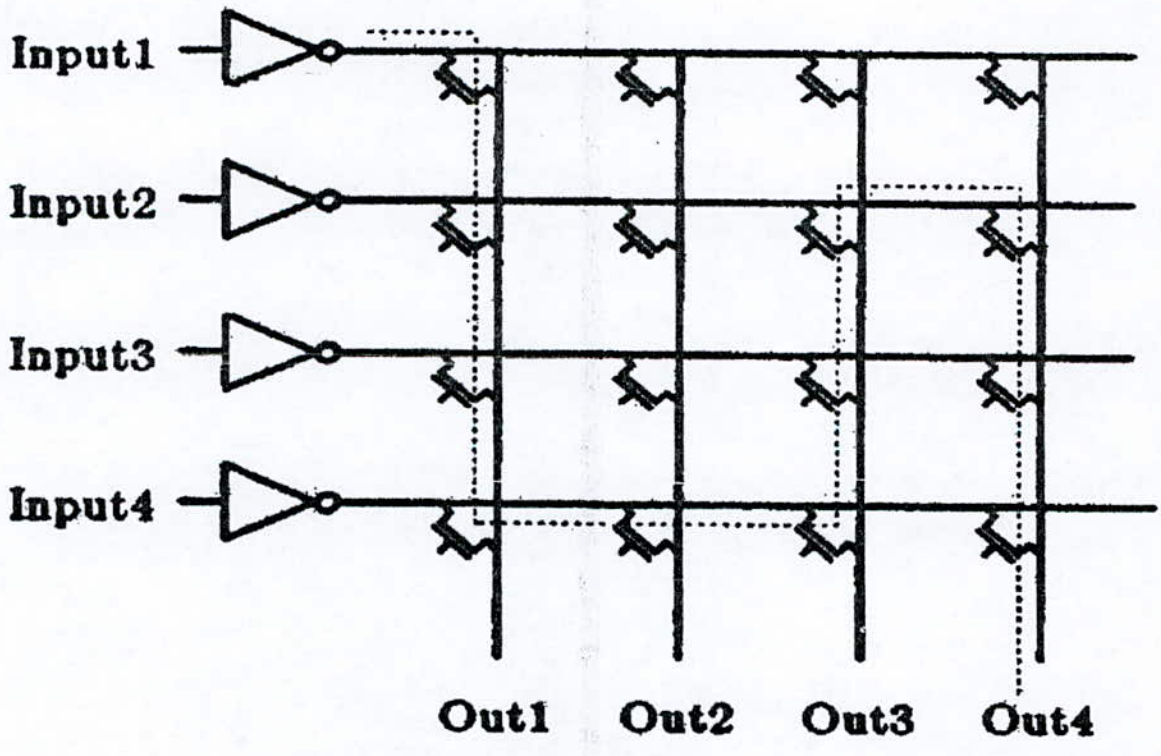


Fig. 1. An exemple of false path

for switch level CAD tools based on path algorithms, is that many false paths (see fig. 1 [3]) can be eliminated. Thus signal flow information leads to reduced analysis time and enhanced accuracy.

In the case of static timing analyzers, signal flow has been recognized as one of the two main source of false paths [15]. Consequently for timing analyzers users, the lack of automatic signal flow determination tools has resulted in the use of manual pruning techniques in order to eliminate false paths, as well as the minimization of the number of element, whose direction is ambiguous in the designed circuit. This forces the designers to either ignore that portion of the design or build behavioral models [16].

Many different types of switch level CAD tools such as timing analyzers [1,2,3], testability analyzers [8], ATPGs [9], functional abstractors [10], and even some simulators [11,12] etc, deal with or try to find signal flow directions. For this purpose, an efficient pre-processing of CMOS networks is necessary in order to identify transistors signal flows.

2 Previous approaches

For the Crystal [3] timing analyzer, signal flows are reported manually during the layout design phase and are propagated through the netlist by a special extractor. This approach however, requires special CAD tools and extensive user intervention. For the TV [2] timing analyzer, a rule based approach is used to determine transistors signal flows. However, this approach is NMOS oriented and is virtually unusable for CMOS circuits. In addition, many rules are "unsafe" and thus lead in certain cases to erroneous results. For the Pearl [4] timing analyzer, an automatic and more general approach is used. It consists of using technology independent and general rules, that are not strictly "safe" though. As a consequence, Pearl has difficulties to cope

with some circuits described in the literature (for example the barrel shifters described in [3], and the RAM cell described in [2]). Furthermore, this approach causes circuit transistors to be processed several times.

In [13], an algorithmic (graph-based) approach is used, in contrast with the previously described rule-based approaches used in [2,3,4]. Despite its relatively better performances, compared to rule-based methods, the running time is still important, especially for circuits containing several bidirectional transistors. Indeed, bidirectional devices need additional treatments which require other identification algorithms. With this approach, as for the rule-based ones, the problem of bidirectional devices has not been solved completely, and then for some "pathological" circuits, unassigned transistors may still exist. Some non static circuits are also not processed correctly. The main disadvantage of this method is that it is too general and does not fully exploit CMOS circuit properties.

Both the rule based and algorithmic approaches have several drawbacks. The principal ones are: pure rule based approaches generally use pattern matching techniques to identify transistors signal flows. These techniques can consider only circuit local effects and thus some rules can be unsafe in certain cases. Besides, these techniques are also time consuming. On the other hand, algorithmic approaches are faster but are less flexible and thus have difficulties to cope with the wide range of CMOS design styles. In addition, bidirectional transistors require to be processed by specialized and additional algorithms, and sometimes need even an exhaustive path search.

An improved version of [13] is presenred in [14]. The system consists of two subsystems: an algorithm-based subsystem, and a rule-based subsystem. With this approach

most circuits are processed correctly for all design styles (even though for precharge logic, precharge nodes are manually specified by the user). However, these two subsystems operate sequentially and do not work closely together, since circuits are first processed by the algorithm-based subsystem, and finally by the rule-based one. Consequently this method suffers from the drawbacks of the pure rule based approaches (ie: use of only local rules that are not strictly safe, use of time consuming pattern matching methods), and some circuits are not assigned a correct direction. Therefore the correctness of the results in all cases as well as the linearity of all the system is not guaranteed.

In this paper[1] , a novel mixed algorithmic and rule-based approach is presented. Its main characteristic is that a unique algorithm is able to process all design styles (static and dynamic, with or without bidirectional transistors) without any restrictions, in a consistent manner. The double algorithmic and rule-based nature of our algorithm, alleviates the drawbacks of both approaches. It permits to avoid the use of the time consuming pattern matching techniques often used with pure rule-based approaches. It also considers global effects due to our algorithmic approach, which enables transistor global circuit informations to be propagated and then gathered in the neighbourhood of the transistors to be identified. This recursive propagation mechanism allows to use global rules that consider path informations, and at the same time avoids an exhaustive path search. As a consequence, the used rules are general and technology independent. Our approach uses also a relatively small number of rules based on: node strength concept, local transistor information and global path informations.

1 This paper is an extended version of [5]

3 Problem formulation and definitions

CMOS networks are modeled in the literature as undirected graphs, where each graph node represents a node circuit and where each graph edge represents a transistor (drain source or channel connection). A network is considered as a set of connected components called transistor groups [1]. For signal flow determination purpose, additional features are required to represent circuits: power sources and input nodes are represented by source nodes. Output nodes as well as nodes with transistor gates are represented by sink nodes.

From a computational point of view, the problem can be stated as follows: Given an indirect graph and some special nodes called sink and source nodes, the objective is to identify the orientation of the directed edges.

Definition 1 [13]: A transistor with two diffusion nodes u and v, can be assigned a direction from u to v, if and only if there exists two simple vertex disjoint paths, one between a source and u, and the other between v and a sink. This transistor is bidirectional if and only if both directions from u to v and from v to u can be assigned.

Definitions 2: In this paper we consider that:

- the connected transistors to a node are those connected by either drain or source (to this node),

- - the exploration of a transistor consists of traversing this transistor from drain to source or from source to drain.

- a neighbour of a node relatively to a transistor is the node which can be reached when traversing this transistor, for example in fig. 1 the neighbour of node B through transistor 1 is the node A.

Definition 3: During network analysis for an unknown transistor, signal flow is considered relatively to one of its diffusion node (generally the node being currently analyzed) as : An input signal if the signal flow goes from the node, and an output signal if the signal flow comes to the node. For example in Fig. 2, signal flow of transistor #3 is an output signal relatively to node C and is an input signal from node D.

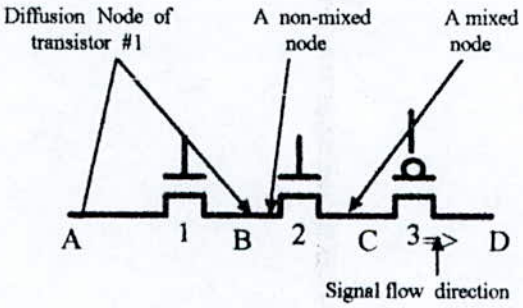


Fig. 2. Illustration of transistor terminology

Definitions 4: Each node has a strength as defined by Bryant [1]. However for signal flow determination, seven strengths are newly defined as follows (in increasing relative strength):

Destination node: any sink node.

Charged node: any node connected by a transistor to a destination, or could be any node connected to a charged node.

Mixed node: is a network node where both PMOS and NMOS transistors are connected by either drain or source, and where ends, at least one sequence of NMOS or PMOS transistors, from only one kind of source (either VDD or GND or a primary input).

Super-mixed node: is a network node where both PMOS and NMOS transistors are connected by either drain or source, and where ends, at least two sequences of only NMOS or PMOS transistors, one from VDD and the other from GND.

Pulled node: any node connected by a transistor to an Input node or could be any node connected to a pulled node.

Super-Pulled node: any node connected by a transistor to an Input node.

Input node: any source.

Note that the term mixed will refer to both mixed and super-mixed strengths.

4 Algorithm Principle

In order to improve the algorithm performance, circuits are considered to be partitioned into three regions : Pullup, Pulldown and intermediate regions. Pullup and Pulldown are considered to be composed only by the same type of transistors (N or P transistors). Each region will be processed separately. In addition, the algorithm can also process correctly circuits where the above assumption is not true.

The algorithm is based on a depth first search [6] approach. Network nodes are processed from source or sink nodes as far as possible in order to find their strength and also in order to reach other sink or source nodes. Two recursive passes are performed: the first one, called the forward pass, is initiated from source nodes and tries to reach source or sink nodes. The second one, called the backward pass, performs the same steps but from sink nodes as long as the current node strength is greater or equal to the neighbour's. The first pass processes pullup and pulldown regions, while the second processes the intermediate region if it exists. Both passes

are event driven with node exploration. Note also that during both passes, if a feedback path is detected then the analysis will simply backtrack from it and explore the other paths (this is due to the definition (see def. 1) of the signal flow that considers simple vertex disjoint paths).

```
Begin
      FOR each group of the circuit
            perform initialisation pass
            perform forward pass(ie forward phase 1 and forward phase 2)
            perform backward pass
      END
END
```

Fig. 3. Overview of the main algorithm

The algorithm assumes that all source and sink nodes of each group are identified and respectively stored into lists called forward and backward lists (that are associated with respectively the forward and the backward passes). This process can be done in linear running time with the number of network nodes and transistors by exploring each group.

4.1 Forward rules

All transistors are first assumed to be bidirectionals. Signal flow is deduced by applying one of the following rules:

1) If a transistor is directly connected to an input node then the signal flow can only be an output signal from the input node.

2) If only one transistor connected to a node is unknown and all other signal flows are input signals, then the unknown transistor flow must be an output signal.

3) Given an analysis direction starting from source nodes reaching an unknown transistor (actually to one of its diffusion node), and also if this transistor is connected to a mixed or destination node, then its flow is an output signal.

4) Given an analysis direction starting from source nodes reaching an unknown transistor, if all paths coming to this transistor (actually to one of its diffusion node) reach only source nodes, then this transistor flow must be an input signal.

5) Given an analysis direction starting from source nodes reaching an unknown transistor, if all paths coming to this transistor reach only mixed nodes or a destination , then this transistor flow must be an output signal.

6) Given an analysis direction starting from sources and reaching an unknown transistor, if all paths coming to this transistor reach source and mixed nodes, thus we consider the neighbour of the current node relatively to the unknown transistor. If from its neighbour all paths reach source and sink nodes then the transistor is bidirectional, otherwise it is unidirectional (transistor flow is an input relatively to the current node). This rule assumes that the neighbour of the current node has been completely explored. If it is not the case, then the current node is locked and the transistor stored into a list and also locked, in order to be processed again when all the remaining transistors will be explored or scaned.

Note that all these rules are safe (rule 1 is trivial, rule 2 is based on Kirchkoff law, rules 3,4,5 and 6 are based on the path definition of the signal flow as well as the path strength concept). Note also that these rules can be split into two categories:The first one called the local rules includes rules 1 and 2. The second one called the global rules includes rules 3,4,5 and 6. Due to the fact that local rules are faster than the global ones, the algorithm first attempts to apply the

local rules as long as possible. When it is no longer possible, then the algorithm tries to apply the global rules.

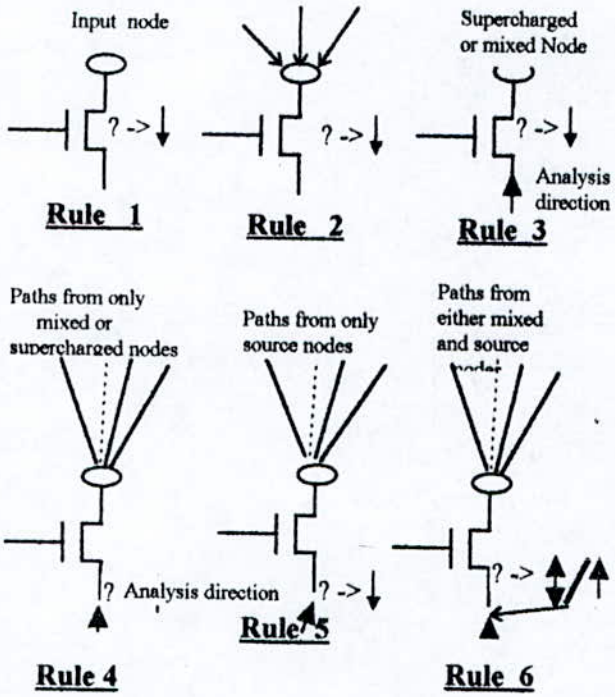


Fig 4: Forward rules

4.2 Backward rules

Note that the backward rules are roughly the duals of the forward ones, except that only the global rules are applied. Therefore, the backward rules consist of only four rules:

7) Given an analysis direction starting from sink nodes reaching an unknown transistor, and also if this transistor is connected to a mixed node with all connected transistors locked, Thus its flow is an output signal.

8) Given an analysis direction starting from sink nodes reaching an unknown transistor, if all paths coming to this transistor reach only sink nodes, thus the same procedure as used in rule 6 is used (test of other neighbour node).

9) Given an analysis direction starting from sink nodes reaching an unknown transistor, if all paths coming to this transistor reach only mixed nodes, thus this transistor flow must be an output signal.

10) given an analysis direction starting from sink nodes and reaching an unknown transistor, if all paths coming to this transistor reach either source and mixed nodes, and if the strength of the current node is greater than the previous explored node (the other diffusion node of the unknown transistor), thus the unknown transistor flow must be an input signal.

5 Algorithm description

5.1 Forward pass

The forward pass comprises two phases. The first phase (FWD-phase1) starts from source nodes of the current group, and considers the transistors that are directly connected to them and applies rule 1. Then starting with the other neighbour node of the identified transistors, the first pass tries to apply rule 2 as long as possible. Typically, this phase will process serial sequences of transistors starting from sources and also reconvergent sequences of serial transistors. During this phase, signal flows are determined directly as long as parallel combinations of transistors are not found. When this happens, the last analyzed node is stored at the end of the forward list and the phase 1 is stopped. If transistors remain not processed from the previous phase then the second phase (FWD-phase2) of the forward pass starts from nodes still stored into the forward list and which have not yet been processed. Then it recursively explores or scans all possible

transistors as long as possible, until it reaches either a source or a mixed or a destination node, then it applies the rule 3. The experience has shown that the first phase is faster than the second one.

During the second phase of the forward pass, transistor identification is done when all possible paths have been explored (ie : during backtracking), the algorithm tries to apply rules 4, 5 or 6. When a transistor signal flow is precisely determined, it is locked (ie: it becomes virtually non-existent). In the same manner, nodes are also locked when all paths have been explored. During exploration, if a sink or source node is encountered, then this information must be propagated to the previous ones. For this purpose within each node record, are provided boolean flags called sinkfound and sourcefound fields. They are updated after each backtrack from a node and therefore path informations are progressively propagated to previous nodes. Note that the second phase is able to process alone the circuit pullup and pulldown regions, and thus the first phase is not mandatory, it is however used as much as possible because of its high speed.

5.2 Backward pass

Once the forward procedure has been applied to a circuit, generally a few transistors remain unprocessed. The backward pass is thus necessary to identify these transistors. This pass can be considered as the dual of the forward one. It uses the backward rules and starts from sink nodes rather than source nodes. It also tries to reach source or sink nodes as long as the current node strength is smaller than pulled. The main difference is that only the second pass is applied, since rules 1 and 2 are canceled. During the second phase, the difference is that the backtracking

starts only when mixed nodes with all connected transistors already explored are encountered. The last difference is that all transistors directly connected to sink nodes are first locked. Their signal flow is determined at the end of the identification process, according to the backward rules. The bidirectional I/O ports are also treated in this pass. They are first considered as normal nodes (non I/O nodes), and then treated during the backtracking. Note that the node stength concept is mainly used during this pass in order to identify the signal flow of conflicting transistors.

5.3 Main procedure

Both procedures (forward and backward passes) must be repeated for all the groups connected to the current one by transistor gates. Then for all connected groups once processed, the procedure is also repeated for all the neighbouring groups, and so on until all circuit transistors are identified. Note that this process is dynamic, and thus integrated with the algorithm.

5.3 Example of application

In order to show the many features of this algorithm, we have selected the following example (see fig. 5) where the transistors are numbered according to their identification order. First the forward phase 1 starts with the source nodes. It applies rule 1 to transistors 1, 2, 3, 4 and 5 which are directly connected to VDD or VSS. It then stores nodes I, J and O in the forward list, and sets their sourcefound flags. Then the analysis can start from one of these nodes, say node I. Rule 2 is applied to transistor 6 during its exploration and the analysis stops, since rule 2 can no longer be applied to node A. The analysis continues from node J that is extracted from the

forward list, and applies rule 2 to transistor 7, and gets again to node A (since transistor 7 is traversed). This time, rule 2 may be used since transistors 6 and 7 have been identified as "in transistors" to node A, and only transistor 8 remains to be identified. According to rule 2, transistor 8 is identified as an "out" transistor from node A, and so is transistor 9. The analysis then reaches node B, and phase 1 stops. During this phase, many circuit informations such as the strength of nodes I,J,O have been found, and others propagated (such as the sourcefound flag of node B which was set to be true, which means that a source can be reached from this node).

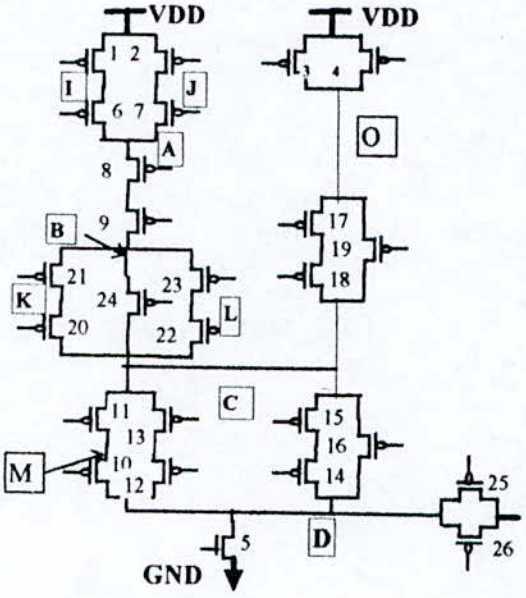


Fig. 5: Example

The recursive phase 2 can then start from nodes B or O which were stored in the forward list (during phase 2, the identification is done during backtracking ie. when all the transistors of a node have been explored), for instance let us consider node B. The analysis can start with either transistor 4, 21 and 23 depending on the order in which they are stored in the circuit data base. However, as we will see the result is the same whatever order is chosen. Assume that transistor

21 is chosen, then the analysis explores transistor 20 and then reaches node C. Subsequently, the algorithm can choose transistors 11,13,15,16,18,19,22 and 24. If it chooses transistor 24 (the order also does not matter), then the analysis gets back to node B which has been flagged "visited". The algorithm starts backtracking and simply returns to node C without any action. It explores transistor 22, and then transistor 23 and gets also to node B which is still set "visited". As previously, the algorithm backtracks and gets to node C, and explores transistor 11, for instance. Then it explores transistor 10 and gets to node D which is a mixed node. Thus, rule 3 is applied and transistor 10 is assigned an input signal flow to node D. The analysis backtracks since it cannot go futher, and propagates to node M the information that a mixed node has been reached. This information stored into the sinkfound flag of node M, enables to identify the signal flow of transistor 11, according to rule 5. This information is also propagated to node C when backtracking from transistor 11. The same steps are performed for transistors 12 and 13, and then to transistors 14 and 15, and finally to transistor 16. All these transistors, according to rule 5, are oriented toward node D. Subsequently, the analysis considers the remaining transistors that are transistors 18 and 19. Assume that transistor 18 is first considered, then transistor 17 is explored and thus node O is reached. Since node O has been previously locked and its sourcefound flag set, then the analysis backtracks and transistor 17 is identified as out of node O according to rule 5 as well as transistor 17. Then transistor 19 is explored and node O is also reached, thus transistor 19 is identified according to rule 5. At this point, all the transistors from node C have been explored, and the algorithm begins to backtrack. At this stage of the analysis, the sourcefound and the sinkfound flags of node C are both set. This means that from node C there is at least one path towards a sink node and at least one path towards a source node. The algorithm backtracks to transistor 20, and applies rule 6 since node K has been completely

explored, and no path to a sink node exists. The algorithm then propagates the information that a sink node has been reached to node K, and according to rule 5 sets the signal flow of transistor 21 towards node C. Next, the algorithm may consider transistor 24, and thus gets to node C. Since this node has been completely explored, it is as a result locked. Therefore, the analysis does not need to go further and is stopped. Since the sourcefound and sinkfound flags of node C are both set, hence rule 6 is applied. But transistor 24 cannot be identified since node B has not been completely explored. Therefore transistor 24 is "delayed", ie. stored in the exploration list in order to be explored again later. The analysis considers transistor 23 and then transistor 22, and gets to node C (which is locked). Here rule 6 can be applied, and thus transistor 22 and then transistor 23 are identified. Transistor 24 is explored again and since all the paths from node B have been explored, it is identified according to rule 6. Since the pullup and pulldown regions were processed, thus the forward pass stops and the backward pass starts. Only transistors 25 and 26 are processed by the recursive backward pass, and as a result these transistors are identified according to rule 9. At this stage all transistors have been identified. Note that transistors 20, 21, 22, 23 were processed two times, and that transistor 24 was processed three times, while all the others (21 transistors) were processed only one time.

7 Algorithm complexity and discussion

Our algorithm has a linear running time, since even for the worst case the transistors are processed a fixed number of times. In the great majority of cases, each transistor is processed only one time (plus one for the initialization phase). However, for highly parallel circuits, transistors are processed twice, once from drain to source and once from source to drain. Occasionally transistors are processed up to three times in certain cases (typically one transistor

per group), when rules 6 and 8 are applied. The linearity of the algorithm is achieved due to the use of flags for network nodes and transistors. Each time a node has been completely explored, or a transistor signal flow precisely identified, then these nodes or transistors are locked avoiding further explorations.

Our mixed approach has the advantages of the rule based one (ie. the flexibility and the ability to cope with the great variety of CMOS design styles) as well as the advantages of the algorithmic approach (ie. fast processing time and the ability to consider circuit global effects). Its main characteristic is that a unique algorithm is able to automatically process all design styles (static and dynamic, with or without bidirectional transistors) without any restrictions. It uses a recursive path propagation mechanism that is event-driven with transistor identification, that permits to avoids an exhaustive path search. Our algorithm does not need additional data structures such as the ST-graph used in [13]. Instead, our algorithm operates directly on circuit data structures, and thus reduces memory consumption and avoid dynamic memory manipulations. Besides, our algorithm does not need any user intervention during circuits analysis.

The other interesting features that contribute to enhance the performance and the accuracy of our algorithm, are:

* the used rules are "safe" general topological rules instead of ad hoc or technology dependent ones, such as "all transistors fed by the output of an inverter are unidirectional" that is used in [2] and that may turn in certain cases to be unsafe.

* a dynamic rule selection mechanism, that enables only a small subset of rules to be applied during a particular phase of the algorithmic analysis.

* the use of the node strength concept (as used in [10] and [12] for signal flow determination) and the notion of global rules (that consider path informations), reduce the number of rules (ten "safe" rules).

* the dynamic circuit partitioning of circuits (actually each group) into three three regions: pullup, pulldown and intermediate regions.

8. Tests

Our algorithm has been applied on a wide range of circuits using different design styles and techniques such as : Static and dynamic circuits. Complementary, ratioed, precharge and pass logics. These circuits are: All adders described in [7], Various barrel shifters, a pass transistor MUX-DEMUX, dynamic and static latches etc. We have also successfully tested all the pathological circuits available in the literature [2,3,13] including the pass transistors arrays that remained a problem for many switch level timing analyzers [3,4], and the tiny XOR well known because of its performances but also because of the problems it causes to switch level CAD tools [7].

In almost all the tested circuits, all the unidirectional and the bidirectionals transistors have been correctly identified. The exceptions are circuits 10 and 15 (see table 1), which contain non-parallel and serial combinaisons of transistors such as the parity generator represented in figure 6. Due to their non conventional design style, three unidirectional transistors were each time left bidirectional. However, for all the other tested circuits all the transistors were precisely identified. In addition, in all cases, our algorithm has never produced false results, since some unidirectional transistors were left bidirectionals, but no bidirectional transistor was left unidirectional.. A possible solution to this problem consists of extending the number of node

strengths, with a substantial running time overhead though, as well as the loss of the algorithm linearity. Since most of the real circuits consist of combinaisons of serial and parrallel transistors, this solution has not been chosen.

The measured CPU times (see table 1) are equal to the overall running time of our software, minus the time needed to process the netlist and to read the library. The tests show an

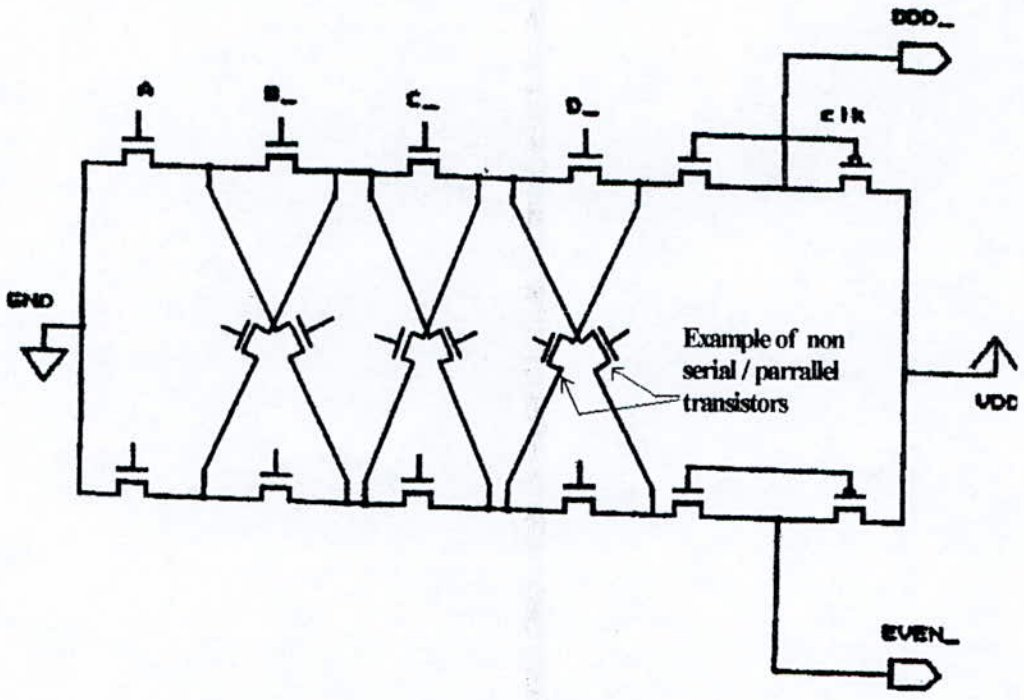


Fig. 6. Example of circuit containing non serial/parrallel transistors

average running time of 40 000 transistors/second on a 25 MHz Mips workstation. The best results have been obtained with static circuits, where the forward phase 1 (which is the fastest phase) was applied to the majority of transistors. The worst results have been obtained with respectively, dynamic and transmission gates circuits, where respectively the forward phase 2 and the backward phase (which are the slowest phases) were applied to the majority of transistors. The comparison with the previous approaches shows relatively good performance (about 330

transistors/sec on VAX11/785 [2], about 2000 tr/sec on Sun 3/60 [13] and between 1000 to 7000 tr/sec on Sun Sparc 2 [14]).

9 Conclusion

We have presented a new approach for signal flow determination of MOS transistors. Its *main* characteristic is that it is based on a mixed rule-based and algorithmic approach. The rule-based aspect of our approach is characterized by the use of relatively few "safe" general topological rules that considers global effects. The algorithmic aspect is on the other hand characterized by the use in conjunction of: The recursive mechanism to propagate global path informations, the node strength concept and the partitioning of circuits into three regions. As a result, the running time is suitable for processing large VLSI CMOS circuits, and at the same time does not need any significant memory overhead. Besides, in contrast to previous approaches, our algorithm is able to correctly and accurately identify most unidirectional and bidirectional transistors in all the tested benchmarks, without any restriction regarding to circuit design style.

Moreover, we believe that the flexibility of our approach should allow to easily adapt our algorithm to new and advanced CMOS design styles or techniques such as BiCMOS.

Circuit	Reference	Comment	number of unidirect. transistors	number of bidirect. transistors	number of unidirect. trans. left as bidirect.	CPU time
1	[7] page 312	32 bits Complementary adder	1216	0	0	21.8 ms
2	[7] page 323	Carry lookahead adder	18	0	0	0.5 ms
3	[7] page 316	32 bits Dynamic adder	704	0	0	19.6 ms
4	[7] page 324	Manchester carry chain	21	0	0	0.6 ms
5	[7] page 173	Transmission gates MUX-DEMUX	16	0	0	0.4 ms
6	[7] page 318	32 bits Transmission gate adder	704	0	0	23.9 ms
7	[3] page 341	Barrel shifter	20	0	0	0.6 ms
8	[3] page 342	Bidirectional Barrel shifter	24	16	0	1.2 ms
9	[7] page 350	Static CMOS RAM cell	6	2	0	0.2 ms
10	[7] page 354	Parity generator	18	0	3	0.6 ms
11	[13] page 495	Circuit with bidirectional I/O ports	14	0	0	0.5 ms
12	[7] page 216	Cross coupled fflop	16	0	0	0.5 ms
13	[7] page 361	Column tree decoder	14	0	0	0.3 ms
14	[7] page 364	Diffenretial Sense amplifier	14	0	0	0.5 ms
15	[7] page 170	CVSL Xor	17	0	3	0.6 ms

Table 1: Tests measured on a 25 Mhz Mips R3000 workstation

REFERENCES

[1] Bryant, A switch-level model and simulator for MOS digital systems, pp 160-177, Feb 1984.

[2] Norman P. Jouppi, Derivation of signal flow direction in MOS VLSI, in IEEE Trans. on CAD vol. 6, No3, pp 480-490, MAY 87.

[3] John K. Ousterhout, A switch-level timing verifier for digital MOS VLSI, in IEEE Trans. on CAD Vol CAD-4 No 3, JULY 85.

[4] James J. Cherry, PEARL a CMOS timing analyzer, in 25 th DAC, pp 148-153, 1988.

[5] A.R. Baba-ali, A. Farrah, An efficient algorithm for signal flow determination in CMOS/VLSI, in proceedings EDTC'96 (European Design & test of Computers), 1996.

[6] R.Tarjan, Depth-first search and linear graph algorithms, in SIAM computer, pp 46-160, JUNE 72.

[7] N.West, K.Eshraghian, Principles of CMOS VLSI design, a system perspective, Addison Wesley, 1985.

[8] M.A. Cirit, Switch level random pattern testability analysis. Design Automation Conf., pp 587-590, 1988.

[9] H.H. Chen et al, An algorithm to generate tests for MOS circuits at the switch level, Int. Test Conf. , pp 304-312, 1985.

[10] M. Boehner, LOGEX: An automatic logic extractor from transistor to gate level for CMOS technology, Design Autom Conf., pp 517-522, 1988.

[11] Z. Barzilai et al, SLS_A a fast switch level simulator,IEEE trans. on CAD, pp 838-849, Aug.1988.

[12] D. Adler, Switch level simulation using dynamic graph algorithms,IEEE trans. on CAD, pp 346-355, March 1991.

[13] K. Lee et al, A New method for assigning signal flow direct.to MOS trans.,in ICCAD pp 492-495,1990.

[14] K. Lee et al, An integrated system for assigning signal flow directions to CMOS transistors, IEEE trans. on CAD, pp 838-849, Aug.1988.

[15] S. Perremans et al, Static timing analysis of dynamically sensitizable paths, in proceedings of the 26 DAC, pp 568-573, 1989.

[16] S. Napper, Static timing analysis- a demanding solution, Electronic Engineering, pp 35-44, January 1995.

Annexe 2

Formal Proof of the Correctness Of an Efficient Algorithm For Signal Flow Determination In Digital CMOS VLSI

A.R. Baba-ali & A. Farah

Abstract

Signal flow determination of CMOS/VLSI digital circuits is a key issue for switch-level CAD tools such as timing and testability analyzers, functional abstractors, ATPGs etc .. and even some simulators. Signal flow determination is used to preprocess circuit MOS transistors in order to improve both the accuracy and the running time of these CAD tools. Existing algorithms can be classified into two main categories: the rule-based approach and the algorithm-based approach. However, both of them have several drawbacks.

In [5] we presented the principle of an efficient algorithm based on a mixed algorithmic and rule-based approach, that overcomes most of the drawbacks of the previous approaches. Besides, our approach compares favorably to the previous approaches, especially for the running time. In this paper, we presents the detailed algorithms, as well as the analysis of their complexity. Moreover, the proofs of their correctness are also presented. It is noteworthy that none of the previous approaches could proove the correctness of their algorithms, and this problem has been recognized in the litterature as «difficult to achieve »[11].

Submited to IEEE Transactions on Computer Aided Design of VLSI Circuits and systems

1 Introduction

The signal flow of a transistor is defined as the direction from signal source to signal destination [2]. This information is used by CAD tools in order to improve both their accuracy and their speed. However, due to the bidirectional nature of MOS transistors, this problem is difficult to solve. Consequently, in the case of static timing analyzers for instance, the signal flow problem has been recognized as one of the two main source of false paths [12]. While for the first source (that is the functional dependencies), extensive studies have been done and efficient solutions exists, for the signal flow problem relatively fewer studies have been done. Besides, for the signal flow determination problem few consistent algorithms have been described in the litterature. Not only they exhibit poor performances, but also none of them have proved their reliability in all cases.

2 Overview of our approach

We have presented in [5], a novel mixed algorithmic and rule-based approach. Its main characteristic is that a unique algorithm is able to process all design styles (static and dynamic, with or without bidirectional transistors) without any restrictions, in a consistent manner. The double algorithmic and rule-based nature of our algorithm, alleviates the drawbacks of both approaches. It permits to avoid the use of the time consuming pattern matching techniques often used with pure rule-based approaches. It uses a recursive path propagation mechanism based on a Depth-first search [6], that is event-driven with transistor identification, that permits to avoids an exhaustive path search. It also considers global effects due to our algorithmic approach, which enables transistor global circuit informations to be propagated and then gathered in the neighbourhood of the transistors to be identified. This recursive propagation mechanism allows

to use rules that consider path informations, and at the same time avoids an exhaustive path search. As a consequence, the used rules are « safe », general and technology independent. Our approach uses also a relatively small number of rules based on: node strength concept, local transistor information and global path informations.

The other interesting features that contributes to enhance the performances and the accuracy of our algorithm, are:

* a dynamic rule selection mechanism, that enable only a small subset of the rules to be applied during a particular phase of the algorithmic analysis.

* the use of the node strength concept (as used in [8][9] for signal flow determination) and the notion of global rules (that consider path informations) reduced the number of the rules (ten "safe" rules).

* the dynamic circuit partitioning of circuits (actually each group) into three regions: pullup, pulldown and intermediate regions.

The result is that the software has a fast processing time, in the average 40000 transistors/CPU second on a 25 Mhz R3000 Mips station, with a linear time complexity. Our algorithm has been applied on a wide range of circuits using different design styles and techniques such as : Static and dynamic circuits. Complementary, ratioed, precharge and pass logics. These circuits are: All adders described in [7], Various barrel shifters, a pass transistor MUX-DEMUX, dynamic and static latches etc. We have also successfully tested all the pathological circuits available in the literature [2,3,10] including the pass transistors arrays that remained a problem for many switch level timing analyzers [3,4], and the tiny XOR well known because of its performances but also because of the problems it causes to switch level CAD tools [7]. In almost all the tested circuits, all the unidirectional and the bidirectionals transistors have

been correctly identified. The exceptions are circuits that contain non parallel and serial combinaisons of transistors such as the parity generator represented in [7]. Due to their non conventional design style, three unidirectional transistors were each time left bidirectional. However, for all the other tested circuits all the transistors were precisely identified Besides in all cases, our algorithm has never produced false results, since some unidirectional transistors were left bidirectionals, but no bidirectional transistor was left unidirectional.. A possible solution to this problem consists of extending the number of node strengths, with a substantial running time overhead though,and also a loss of the linearity. Since most of the real circuits are composed with combinaisons of serial and parrallel transistors, this solution has not been choosen.

3 Correctness of the algorithm

The algorithm is based on a depth first search [6] approach. Network nodes are recursively processed from source or sink nodes as far as possible, in order to find their strength and also in order to reach other sink or source nodes. Two passes are performed: the first one, called the forward pass, is initiated from source nodes and tries to reach source or sink nodes. This pass is divided into two phases namely: the forward phase1 and the forward phase 2. The second pass, called the backward pass, performs the same steps but from sink nodes as long as the current node strength is greater or equal to the neighbour's. Both passes are event driven with node exploration.

```
( 1)   store in forward queue all transistors of the current group connected to source
or input nodes
( 2)   FOR each stored transistor t
( 3)          set cn as the neighbour of the source node
( 4)          apply rule 1 for the current transistor
( 5)          lock transistor t and node cn
( 6)          store cn in the forward list
( 7)          set source flag of cn
       END FOR
( 8)   WHILE forward list not empty
( 9)          remove the first node from the forward list and store it as cn
(10)          IF current.NBTR - current.NBIN = 1  THEN
(11)          recursively apply phase 1
(12)          ELSE  IF cn not mixed
(13)                FOR each neighbour of cn
(14)                      IF traversed transistor t locked THEN continue
(15)                      set visited cn
(16)                      apply recursively phase 2
(17)                      reset visited cn
                  END FOR
            END IF
       END WHILE
```

Figure 1: main procedure

The correctness of the proposed algorithm is proved through the following propositions.

3.1 Theorem 1

The Forward phase1 identifies the signal flow of serial sequences and reconvergent paths of transistors which start from source nodes, regardless of the exploration order.

```
(1)    IF type of current node = mixed OR sink THEN
(2)          stop recursive exploration
       END IF
(3)    find the NOT locked transistor which is connected to current and and its other
neighbour node n
(4)    lock this transistor
(5)    mark its signal flow as out
(6)    mark its channel node as a source
(7)    IF current.NBTR - current.NBIN = 1  THEN
(8)          call recursively FWD phase 1 with arguments t,n
       END IF
```

Remark: All signal flows are set relatively to the current node.

Figure 2: algorithm FWD phase1
(called with the current transistor and the current node)

Proof : Let us prove it by induction on the transistor number. The basis of the induction (only one transistor connected to a source) is true due to the fact that all transistors connected to source nodes are first identified and processed during the initialisation phase (according to rule 1). Assume that the induction hypothesis is true for n-1 transistors, thus these n-1 transistors of the sequence are already identified. We must then consider two cases:

case 1:

We have a linear sequence of transistors, and when we get to the n-1th transistor, we identify it and then we increase number_in counter (which counts the number of transistors that have an IN signal flow) of the current node by one. Then FWD_Phase1 is called recursively, because the test of line 7 (see figure 2, ie: number of transistors - number_in = 1) is true. Since FWD_phase1 is called with the nth transistor (see figure 3), it is therefore identified and marked.

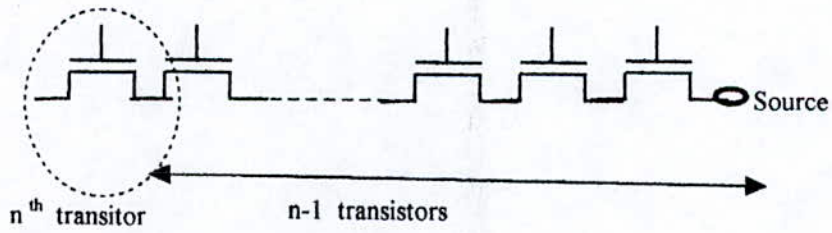


Figure 3: Signal identification of a sequence of transistors

case 2:

We have in this case, a reconvergent sequences of transistors. The proof is also based on induction. The basis of the induction is true, because it represents the case where there is only one reconvergent path, that is a linear sequence of transistors. This case corresponds to case 1 which has been previously demonstrated. For the inductive step we consider that n-1 serial sequences (starting from sources) have already been processed. When the n^{th} path is considered number_in counter of X (see figure 4) node is equal to n-1, thus when the last node of the n^{th} path is processed, its number_in counter is equal to n. As a consequence, the test of line 7 (see figure 2) is true and thus FWD_Phase1 starts processing path X. The exploration order does not affect the result, due to the commutativity of the addition which is used to update the number_in counter .

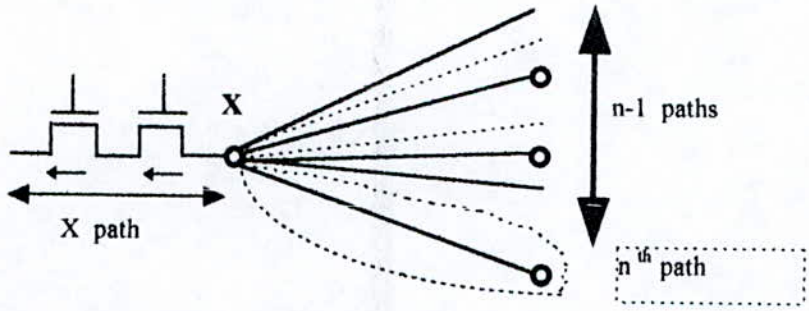


Figure 4: Signal identification of reconvergent paths

3.2 Lemma 1

The FWD_Phase2 explores the remaining transistors (not identified by FWD_phase1) in pullup and pulldown regions, and propagates during backtracking if a source and/or a sink has been found, regardless of the exploration order.

Proof : The proof is by induction on the number of explored transistors. Initially the exploration is performed without identification, while the identification is done during backtracking. The basis of the induction is true since during exploration, transistors directly connected to either sink nodes or sources, are first identified in line 10 (see figure 5). This information is propagated to previously explored nodes (the other channel node of the current transistor) in line 11 (see figure 5). The algorithm starts backtracking when either a sink or source node has been found, or when all paths from a node have been explored. Let us consider as induction hypothesis that for n-1 transistors the sinkfound and sourcefound flags have been propagated (see figure 6). In this case the sinkfound and sourcefound flags of node X are true if there is at least one path from x to a sink or a source. The propagation is done during backtracking (which is done when all paths have been explored) in line 11. It consists of computing the sinkfound and sourcefound according to the following boolean formulas:

Y.sinkfound = Y.sinkfound OR X.sinkfound

Y.sourcefound = Y.sourcefound OR X.sourcefound

For node Y, path informations are correctly propagated through the n th transistor. Thus, all path informations are progressivly propagated to all the other nodes. The exploration order does not affect the result, due to the commutativity of the OR operation which is used to compute the sinkfound and sourcefound flags.

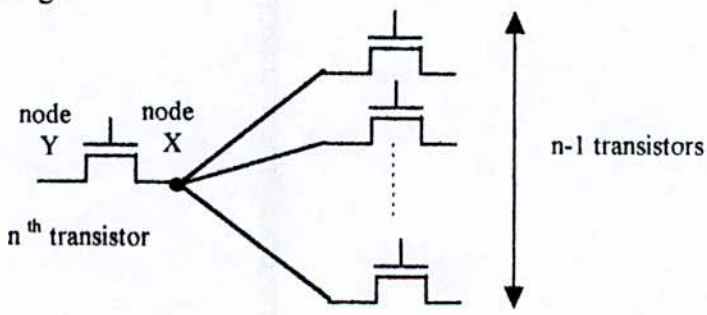


Figure 6: Illustration of the path informations propagation mechanism

```
( 1)   IF type of current node= mixed OR sink THEN
( 2)          mark signal flow of current transistor as out
( 3)          lock current transistor
       ELSE
( 4)      IF current transistor not delayed THEN
( 5)          FOR each transistor t not locked that is connected to the current node
                    (let n the other channel node of t)
( 6)                IF current node visited THEN continue
( 7)                IF current node locked THEN
( 8)                      lock transistor t
( 9)                      IF n type is source or sink THEN
(10)                            mark t signal flow according to n type
(11)                            propagate this information to current node
                          END
(12)                      continue
                    END
(13)                set visited flag of n
(14)                call recursivly FWD phase 2 with t and n
(15)                reset visited flag of n
(16)                propagate sink and/or source informations to previous nodes

            END FOR
          END IF

       ==================     BACKTRACKING    ====================

(17)          IF current node sink flag= TRUE OR source flag =TRUE THEN lock node
(18)          IF sink flag = TRUE AND source flag = FALSE  THEN
(19)                identify current transistor as "in"
(20)                lock current transistor
(21)          ELSE IF sink flag = FALSE AND source flag = TRUE  THEN
(22)                identify current transistor as "out"
(23)                lock current transistor
(24)          ELSE IF sink flag = TRUE AND source flag = TRUE    THEN
(25)                IF previous node is completly explored  THEN
(26)                      lock node and transistor
(27)                      IF not sink flag of previous transitor OR
                               previous.strength > current.strength THEN
(28)                            mark signal flow of transistor as in
(29)                      ELSE  IF previous.strength < current.strength
(30)                            mark signal flow of transistor as out
                          END
(31)                      delay current transistor and node, and mark it as delayed
                    END IF
            END
       END
```

Figure 5: algorithm FWD phase 2
(called with arguments current transistor and current node)

3.3 Theorem 2

All transistors identified by the forward pass are correct.

Proof : According to the definition of signal flows, a transistor with channel nodes (u,v) is unidirectional (say from u to v) if there is a path from a source toward u, and a path from v to a sink. A transistor is bidirectional if this definition holds in two directions.

Assume that the theorem is not true. This means that one transistor at least has not been identified correctly. Consequently there are two possible cases (see figures 7.a and 7.b):

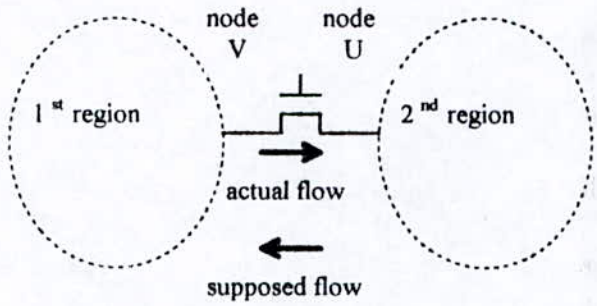


Figure 7.a: type 1 signal flow

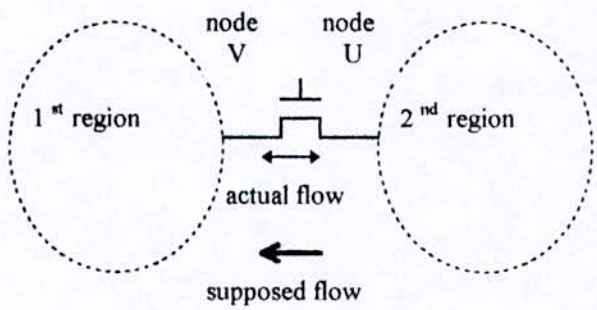


Figure 7.b: type 2 signal flow

Case 1: A unidirectional transistor that has been identified in the opposite direction to the actual direction.

Case 2: A bidirectional transistor that has been identified in one direction. (note that a unidirectional transistor left as bidirectional is not considered as an error).

In both cases, the identification is done using one of the forward rules. Therefore, the error must have been caused by at least one rule. Let us consider first the forward rules (for backward rules the same approach can be used):

3.3.1 Rule 1:

With rule 1, the signal flow is identified from the source to the other channel node. If we have an error with this rule, the actual signal flow would be in the opposite direction (see fig 8). We then get to a contradiction (since this situation is in contradiction with the definition of a source).

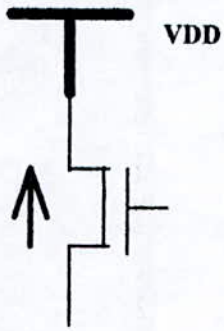


Figure 8: Eventual error with rule 1

3.3.2 Rule 2:

According to rule 2, transistor signal flow is found flowing out of the current node. Assume that we have an error with this rule. The actual signal flow would then be in the opposite direction (see fig 9). Yet again,we get to a contradiction, since this situation is contradicting Kirshkoff's law.

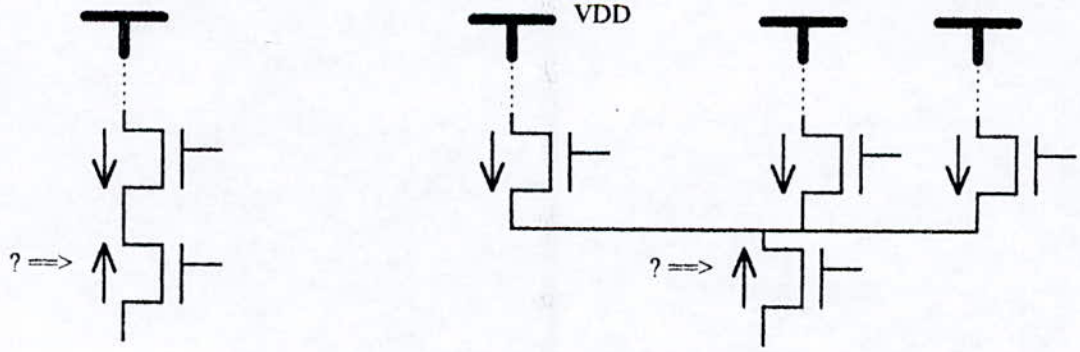


Figure 9: Eventual error with rule 2

3.3.3 Rule 3:

According to rule 3, transistor signal flow is found flowing out of mixed nodes which are the first ones found when we start from source nodes. These nodes are generally separating pullup and pulldown regions. Assume we have an error due to this rule, thus the actual signal flow would be in the opposite direction (see fig 10). Such a situation would imply that there is a source somewhere in the opposite direction. Generally, with correct CMOS circuits, pure pulled regions are composed of one type of transistors, because signals that transit through both NMOS and PMOS are degraded.

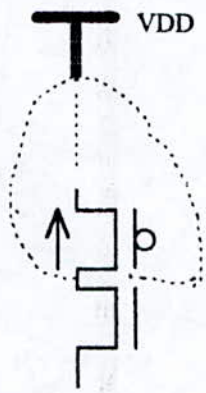


Figure 10: Eventual error with rule 3

3.3.4 Rule 4:

For this case as for the following ones, the identification is done according to the path informations propagated by the forward phases. Note that these path informations are correct according to lemma 1 and theorem 1.

According to rule 4, if we have encountered only a source during backtracking, current transistor signal flow is marked as out of the current node. Assume we have an error with this rule, the actual signal flow would then be in the opposite direction (see fig 11). This means that

there is a path starting from the first region going to the second region where there is only one source. Since there is no sink in the second region, this situation is contradicting the definition of a unidirectional flow (ie: we have a path that ends in a source).

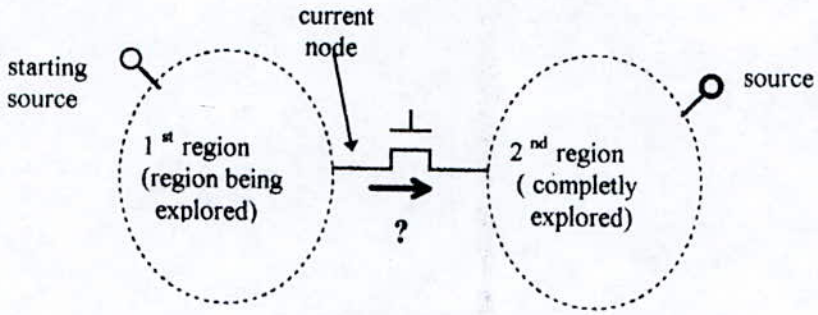


Figure 11: Eventual error with rule 4

3.3.5 Rule 5:

According to rule 5, if we have encountered only a sink during backtracking, the flow of the current transistor is into the current node. Assume, we have an error with this rule, thus the actual signal flow would be in the opposite direction (see fig 12). This situation is also contradicting the definition of a unidirectional flow (we have a path that starts in a sink).

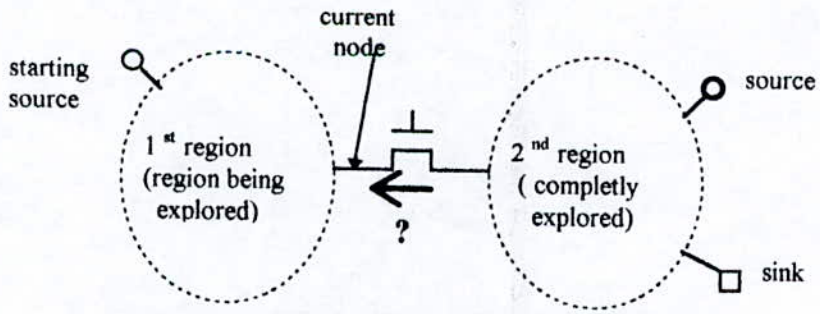


Figure 12: Eventual error with rule 5

3.3.6 Rule 6:

With rule 6 (we have found both a source and a sink), we have two cases:

Case a:

For the first case, the identification can be done only if the first region has been completly explored (in this case the current transistor is delayed). if we don't have a sink in the first region, the flow is identified as going into the current node. If we have an error with this part of rule 6, the actual signal flow could be into the opposite direction (see fig 13). Since there is no sink in the first region, therefore this situation is contradicting the definition of a unidirectional flow.

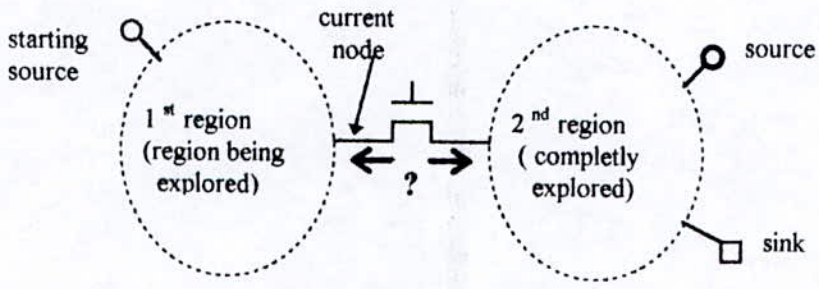


Figure 13: Eventual error with rule 6

Case b:

For the second case, the identification is done according to the node strength of the channel node of the current transistor. If they are equal, the transistor is left bidirectional, in this case no error is possible. In fact, assume we have an error, then the actual signal flow would be unidirectional in one of the two possible directions. In this case, we consider that there is no error since a bidirectional flow includes both cases. However, if channel nodes strengths are different, the signal flow is set from the strongest to the weakest one. If there is an error here, the signal flow is in the opposite direction (ie: from the weakest to the strongest) (see fig.14). Due to the path

blocking principle [1], it means that a stronger path exists since the transistor signal flow has been identified. Then we get to a contradiction.

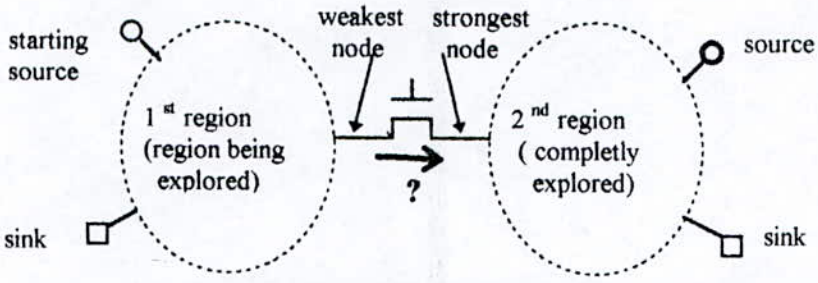


Figure 14: Eventual error with rule 6

3.3.7 Node strengths

We assumed so far that no error could occur due to the node strengths. In other words, we assumed that any path starting in a source could contain only nodes with decreasing node strangths. The node strength concept is used in our context as a mesure of proximity to either sources or sinks, in order to model the « path strengths » which are an approximation of the path resistance. These strengths are summarized as follow:

Source :
Super pulled : a node connected to a source with only one transistor
Pulled : a node connected to a source with a sequence of transistors
Normal : an intermediate strength between pulled and normal
Charged : a node connected to a sink with only one transistor
Super charged : a node connected to a sink with a sequence of transistors
Destination : a sink

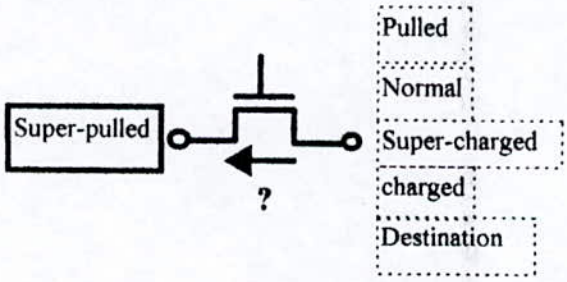


Figure 15: Eventual errors with for instance the super-pulled strength

These strengths represents the stages of a typical path, which starts from a source, reachs a super-pulled and/or a pulled node, gets to a normal node, subsequently reachs a charged and/or super-charged node, and finally reachs a terminal. In order to guarantee the reliability of our algorithm, we have chosen to mark as bidirectional the transistors where the node strength of their diffusion nodes are equal to:

Super pulled	and	Pulled
Pulled	and	Normal
Normal	and	Supercharged
Charged	and	Supercharged
Supercharged	and	Destination

These cases represent the case where node strengths differ only with one level, and which represent to our opinion the critical cases, since these cases could be sensitive to circuit semantics such as transistors dimensions.

Lemma 2

The Backward pass explores the transistors not explored by forward pass, and propagates during backtracking if a source and/or a sink has been found.

Proof : Similar to the proof of lemma 1.

Theorem 3

All transistors identified by the backward pass are correct.

Proof : Similar to the proof of theorem 2.

7 Algorithm complexity

Our algorithm has a linear running time, since even for the worst case the transistors are processed a fixed number of times. In the great majority of the cases each transistor is processed only one time (plus one for the initialization phase). However, for highly parallel circuits, transistors are processed two times, once from drain to source and once from source to drain. Scarcely, transistors are processed up to three times idue to the « delay » mechanism, when rules 6 and 8 are applied . The linearity of the algorithm is achieved due to the use of flags for network nodes and transistors. Each time a node has been completely explored, or a transistor signal flow precisely identified, then these nodes or transistors are locked avoiding other explorations.

Our algorithm consists of three different phases namely : the Forward-phase1, Forward-phase2 and the Backward phase. Each phase processes a disjoint part of the entire circuit. Consequently, the study of the algorithm complexity will be done for each phase.

7.1 Lemma 3

The Forward-phase1 processes transistors at most once.

Proof : This phase typically processes serial sequences (or paths) of transistors, as well as reconvergent paths (see fig. 4). It starts from transistors connected to sources and proceeds as long as it encounters either a sequence of transistors, or a reconvergent path of transistors with only one sequence not processed. This is achieved due to the test of line 10, and line 7

(see figure 2).

In order to prove the linearity of this phase, assume that there is a transistor which is processed at least two times. When the Forward-phase1 is called for the first time with let's say transistor t, this transistor is choosed in line 3, and subsequently in line 4 (see figure 2) it is locked. When Forward-phase1 is called the second time with transistor t, In line 3 transistor t is not choosed because it is already locked. Therefore we get to a contradiction.

7.2 Lemme 4

The Forward-phase2 never allows a transistor to be traversed twice in the same direction.

Proof : The Forward-phase2 performs a recursive Depth-First Search which enables to visit all the transistors of the current pulled region that has not been explored by the Forward-phase1. Starting from the current node, all the connected transistors are explored in the loop of line 5 (see figure 5), and so on until processing all unexplored nodes. Cycling is avoided due to the test of line 6. The proof of the linearity is similar to the previous one. If there is a transistor that is processed at least two times, then it is first selected in line 5, and both its drain and source nodes are marked as « visited » in line 13. Subsequently when the transistor is choosed a second time, it would mean that the test of line 6 is not true. This is impossible, since when a transistor is selected both its drain and source nodes are marked « visited » in line 13.

7.3 Lemma 6

The delay mechanism of the Forward-phase2 never allows a transistor to be traversed twice in the same direction.

Proof : The « delay » mechanism is used in order to « delay » the decision of identifying the current transistor being analyzed. This is due to the fact that all the necessary informations are not yet available. In other words, during backtracking for the current transistor if at least two paths exists toward a source and a sink, then no decision may be made for t until we have explored all paths from its other neighbor node (see figure 5, line 27). This is done by scheduling transistor t in order to be considered when all the other paths have been explored. The proof of the linearity is similar to the previous one. If there is a transistor which is processed at least two times, then it is first delayed in line 31 (see figure 5). Subsequently, when all the possible paths have been explored, the Forward-phase2 is called again with the delayed transistor. The test of line 4 avoids a new exploration, and the backtracking starts immediatly in line 17. Then according to the source and sink flags of the current node, the transistor is identified. In order to be delayed a second time, the test of line 25 must be true. Thus the previous node of the current transistor must not be completly explored. Consequently we get to a contradiction, because in order to identify a delayed transistor, we have to explore first all the paths from previous nodes.

7.4 Theorem 4 :

The Forward phase, has a linear running time.

Proof : Since the Forward phase consists of the following phases: the Forward-phase1 and

Forward-phase2. Besides, since according to lemmas 4,5 and 6 these phases traverse each transistor at most two times, therefore the Forward phase has a linear running time.

7.5 Theorem 5

The Backward phase, has a linear running time.

Proof : Similar to the proof of theorem 2.

9 Conclusion

We have presented a new approach for signal flow determination of MOS transistors, based on a mixed rule-based and algorithmic approach. Our approach is suitable for processing large VLSI CMOS circuits, and at the same time does not need any significant memory overhead. Besides, our algorithm is able to correctly and accurately identify most unidirectional and bidirectional transistors in all the tested benchmarks, without any restriction regarding to the design style. The formal aspect of the proof of the algorithm has stated that some unidirectional transistors may be left bidirectionals, but no bidirectional transistor is left unidirectional. Besides in contrast to the previous approaches, a proof of the correctness of the algorithms has been established.

REFERENCES

[1] Bryant, , A switch-level model and simulator for MOS digital systems, pp 160-177, Feb 1984.

[2] Norman P. Jouppi, Derivation of signal flow direction in MOS VLSI, in IEEE Trans. On CAD vol. 6, No3, pp 480-490, MAY 87.

[3] John K. Ousterhout, A switch-level timing verifier for digital MOS VLSI, in IEEE Trans.on CAD Vol CAD-4, No 3, JULY 85.

[4] James J. Cherry, PEARL a CMOS timing analyzer, in 25 th DAC, pp 148-153, 1988.

[5] A.R. Baba-ali, et al, An efficient algorithm for signal flow determination in CMOS /LSI, in proceedings IEEE ED&TC'96 (European Design & Test of Computers), 1996.

[6] R.Tarjan, Depth-first search and linear graph algorithms, in SIAM computer, pp 146-160, JUNE 72.

[7] N.West, K.Eshraghian, Principles of CMOS VLSI design, a system perspective, Addison Wesley, 1985.

[8] M. Boehner, LOGEX: An automatic logic extractor from transistor to gate level for CMOS technology, Design Autom Conf., pp 517-522, 1988.

[9] D. Adler, Switch level simulation using dynamic graph algorithms,IEEE trans. on CAD, pp 346-355, March 1991.

[10] K. Lee et al, A New method for assigning signal flow direct.to MOS trans., in ICCAD pp 492-495,1990.

[11] K. Lee et al, An integrated system for assigning signal flow directions to CMOS transistors, IEEE trans. on CAD, pp 838-849, Aug.1988.

[12] S. Perremans et al, Static timing analysis of dynamically sensitizable paths, in proceedings of the 26 DAC, pp 568-573, 1989.

Annexe 3

A Delay Optimization CAD Tool For Mixed CMOS/BiCMOS Standard Cells Circuits

A.R. Baba-ali and A. Bellaouar

Abstract

This paper reports on the implementation of a CAD tool that optimizes the delay of logic networks with minimum area penalty. The optimization is done by performing an optimal selection of mixed CMOS/BiCMOS standard cells in VLSI semicustomcircuit design, based on the output load of logic gates. The major problem of this approach, is that the output load is changed by the cell selection and replacement process. Our algorithm alleviates this problem by performing a breadth first search exploration of circuits, starting from their outputs toward their intputs. This approach leads to fast running time (less than 0.1 ms/gate) with a linear time complexity of the algorithm. Moreover, unlike many other tools, our algorithm assumes no restrictions on logic design styles, such as feed-back loop free circuits or only single output cells.

Published in AJSE (Arabian Journal of Science & Engineering), special issue on Microelectronics, October 1994

1 Introduction

One of the key techniques to improve the performance of VLSI systems, is the optimization of logic blocks composing these systems, in terms of speed and area. When using standard cells approach, the optimization can be done by an optimal selection of function cells from a cell library. This type of library is composed of different cell versions that have different templates [1] or sizes [2] in the same technology (CMOS).

In this paper we rather examine the possibility to choose among different technologies such as CMOS and BiCMOS. However, in this first version of our tool, we restrict the used library to only two cell instances of each gate. The process aims to improve circuit performance by making for each gate, a choice between CMOS or BiCMOS cells depending on their load capacitances. This choice is made in order to take advantage of their qualities: namely high-speed and high driving capabilities of BiCMOS, and regularity and low power consumption of CMOS. For that purpose, a CAD tool has been developed to optimize delay of logic blocks that constitute VLSI circuits in order to make them as fast as possible. This tool operates at the logic level, and performs a local(1) non constrained optimization. It uses a mixed CMOS/BiCMOS standard cells library.

(1) The term "local optimization" is used in this paper (gate level context), with the same meaning as in [3] in boolean minimization context (e.g. :"local optimization refers to operations performed on a single node in the Boolean network, or locally in the surrounding neighborhood of the node").

2 Gate Selection

BiCMOS technology is very attractive for ASIC's for the reason of high speed compared to CMOS technology [4]. However this advantage is effective for a gate when its capacitive load exceeds a given threshold. For example, Fig. 1 shows the advantage of BiCMOS technology compared to CMOS in terms of delay. This comparison is made using a 0.8 um BiCMOS technology for the NAND gates with approximatively the same input capacitance. The comparison of CMOS/BiCMOS gate delays driving a capacitive load C shows that the two curves intersect at the crossover point Cx. When the load C is smaller than Cx the CMOS outperforms BiCMOS. But when C is greater than Cx, then BiCMOS is faster. The crosspoint Cx has a different value for each cell and for each technology. Cx is in the range of 0.1-0.2 pF which is equivalent to a fanout of 2-4. Typically for high load BiCMOS gates are 1.5-2 times faster than their respective CMOS version, and dissipate 20% more power. On the other hand CMOS gates have 3-5 times smaller area.

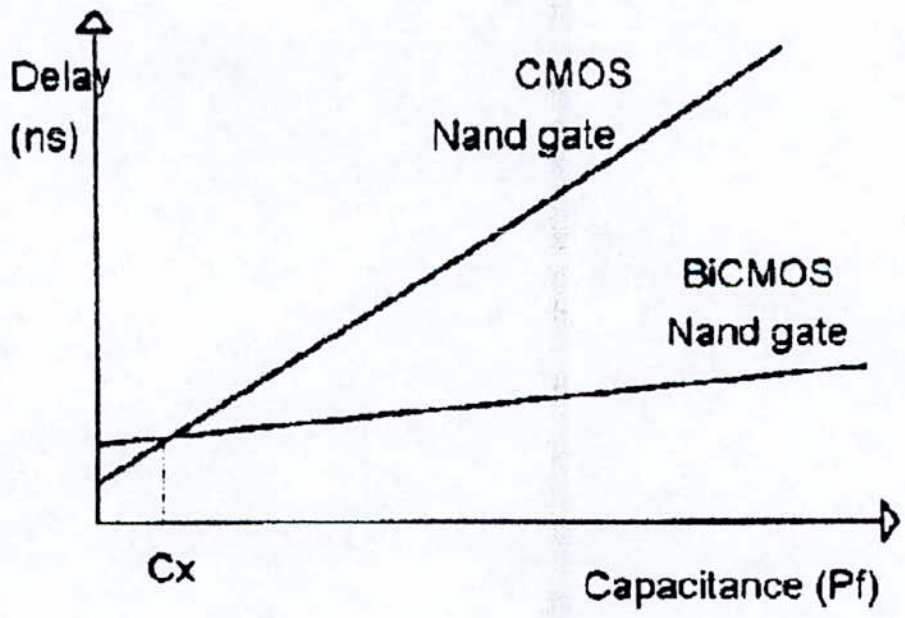


Figure 1. Delay versus *Load.*

In order to achieve high speed operation with minimum area penalty, a strategy for CMOS/BiCMOS cells library design similar to the one proposed by [5] has been adopted: within the used standard-cell library, each cell is available in two versions for the same logic. The first one is designed in BiCMOS technology while the second is designed in CMOS technology. BiCMOS cells are designed to drive a high capacitive load, while CMOS cells are designed to drive a low capacitive load.

The problem in optimizing a logic block using the strategy defined above for CMOS/BiCMOS library design, is to choose for each logic gate its appropriate cell version in the library. The selection technique used to choose a cell within a logic block is based on the following rules:

1) Initially, all cells are considered by default as CMOS in order to increase circuit density.

2) All output cells are replaced by BiCMOS cells because the inter-block wiring capacitance is assumed large.

3) The remaining cells are selected by comparing the value of their respective output load capacitance to the corresponding crossover capacitance Cx. This load is equal to the sum of all input capacitances of the driven cells and the interconnection load. The latter is not taken into account by the first version of our tool because it is not yet known at logic design level. When the load is evaluated, it is compared to Cx in order to select the appropriate CMOS or equivalent BiCMOS logic cell.

3 Selection Algorithm

3.1 Basic principle

In general the load capacitance of a cell depends on the type of the driven cells, since the input capacitance of a cell can be different according to the fact that a cell is either CMOS or BiCMOS. Therefore, before selecting the type of a given cell, all the cells driven by this cell must have already been selected (Fig.2). This fact implies that for a complete circuit, cells must be processed starting from the outputs, selecting their driving cells, and then selecting cells connected to them, and so on until reaching the input terminals.

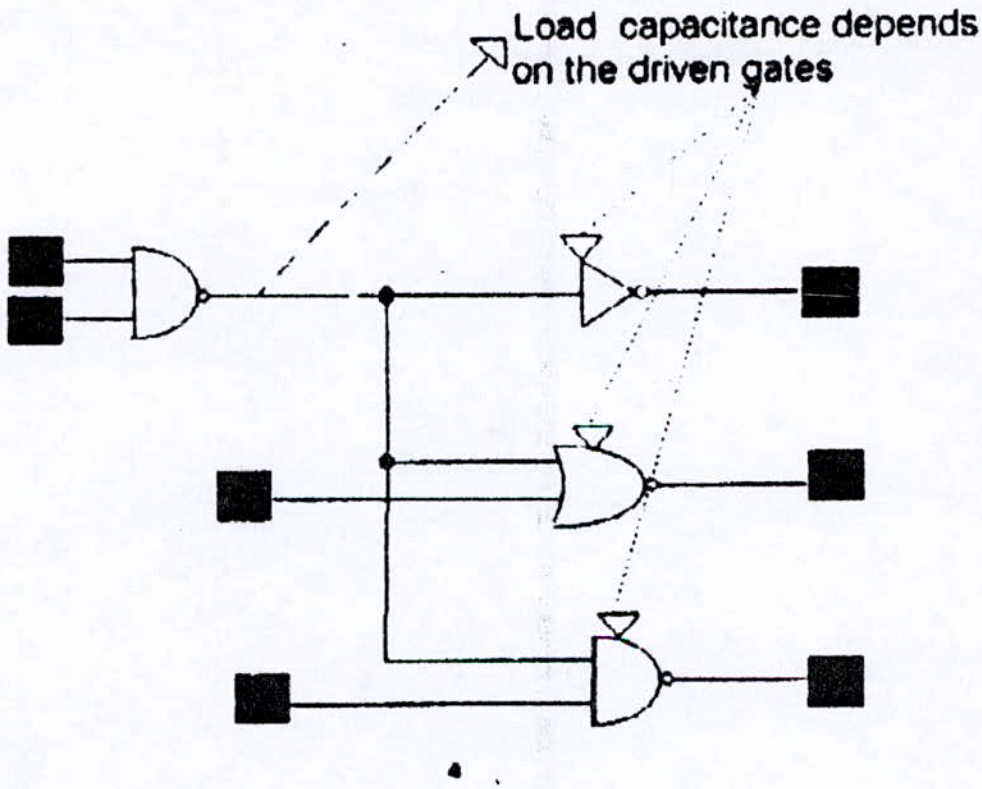


Figure 2. Example of Gate Dependency.

The natural way to achieve this is by using the breadth first search (BFS) graph algorithm [6]. BFS algorithm successively selects cells according to their decreasing logical level as

shown in Fig. 3, in a way similar to the one used by the POLARIS [7] mapping system for polarity propagation (without feed-back loop elimination though). The BFS algorithm associates with each cell a counter which represents the number of driven cells. Each time a cell is selected, then the counters of all cells driving it are decremented. Thus a cell can be selected only if its associated counter is equal to zero, which means that all the driven cells have been processed.

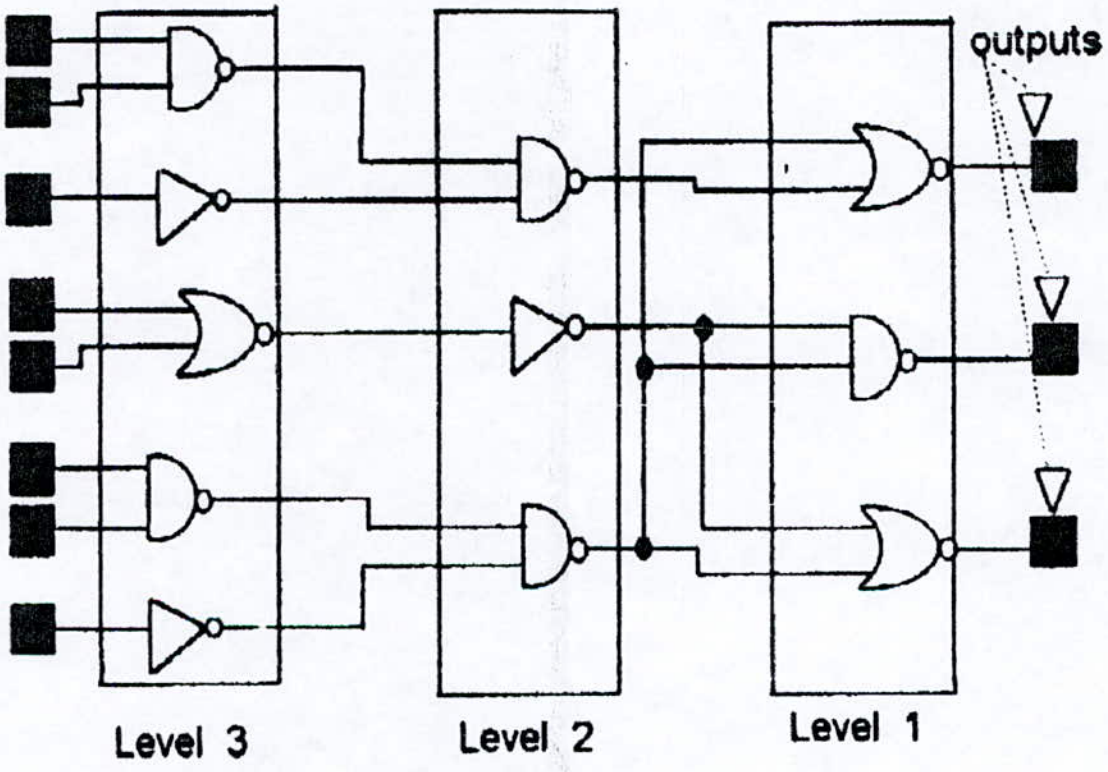


Figure 3. Example of Processing Order.

Some modifications must be performed to the original BFS graph algorithm in order to take into account the fact that a cell can have many outputs. Note that a circuit at the gate level is modeled as a graph (actually an hypergraph), where each node represents a cell and an edge represents a net. With pure BFS a predecessor or successor counter (depending on the forward or the backward direction of the analysis) is used. However, for our purpose we rather use two counters. The first one that is associated with graph nodes, represents the number of cell outputs. While the second that is associated with the edges, represents the number of driven cells. A cell with many outputs is selected when all its outputs have been reached and their respective load capacitances computed. In this case, the cell is selected as BiCMOS if at least one of its outputs has a load capacitance greater than Cx.

```
IDENTIFY()
 1  enqueue( all circuit primary outputs)
 2   WHILE not queue_empty()
 3        current_node=dequeue()
 4        current_cell=driving_cell_of[current_node]
 5        IF capacity_of[current_node] > Cx_of[current_cell]
 6        THEN type_of[current_cell]=BiCMOS
 7        ELSE IF type_of[current_cell] <> BiCMOS
 8             type_of[current_cell]=CMOS
 9        decrement output_counter_of[current_cell]
10        IF ouput_counter_of[current_cell] <> 0
11        THEN continue
12        FOR each_input_of[current_cell]
13           current_cell=next_input[current_cell]
14           IF gate_counter_of[current_cell] = 0
15           THEN continue
16           decrement gate_counter_of[current_input]
17           update_capacity_of[current_input] according to
                  type_of[current_cell]
18           IF gate_counter_of[current_input] = 0 AND
                  type_of[current_input] <> Primary Input
19           THEN enqueue(current_input)
```

Fig 4 CMOS/BiCMOS identification algorithm

3.2 Feed-back Loops Detection

Unfortunately digital circuits may contain feed-back loops such as cross coupled NAND gates that complicates the algorithm. Fig. 5(a) shows that gate #3 cannot be selected until gate #2 is selected, because gate #3 drives gate #2. While gate #2 cannot be selected until gate #3 is selected, because gate #2 also drives gate #3. The solution we have adopted to solve the feed-back loops problem is as follows. First all the feed-back loops are identified. Then the capacitance and the value of the counter associated with the feed-back node (node (b) as illustrated in the example of Fig. 4) are updated.

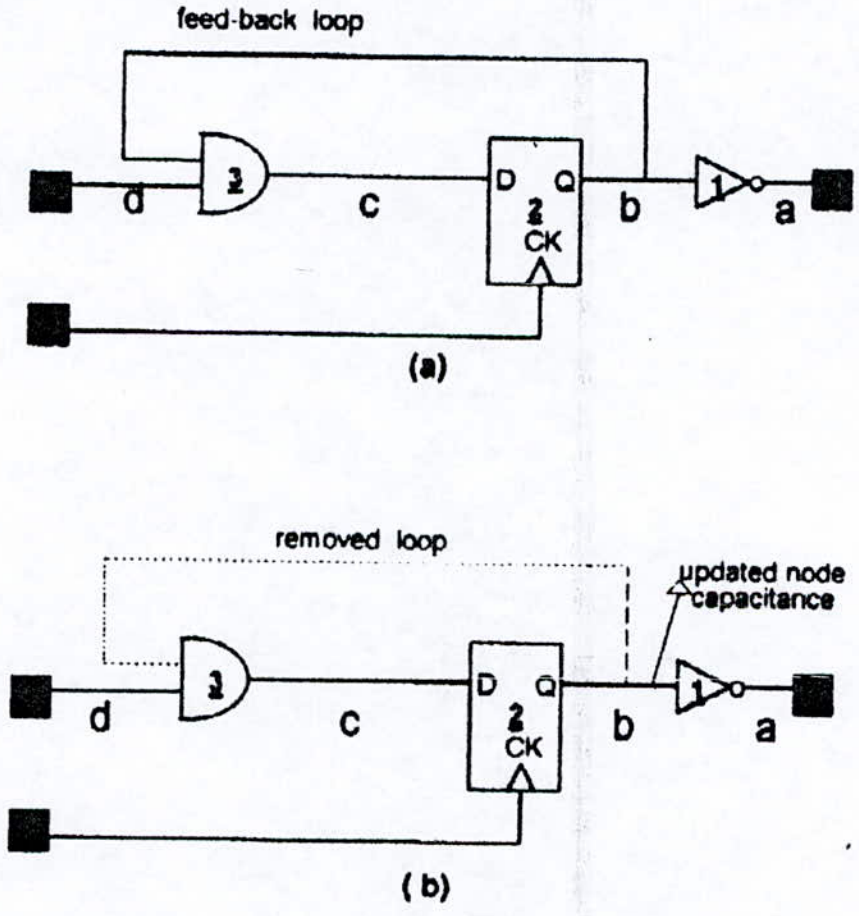


Figure 5. Circuit with a Feed-Back Loop.

The net counter is decremented, and the capacitance of the node is increased with the maximum value of the input capacitance. For example in Fig. 5 when the feedback loop b -> c -> b is found, we choose the maximum input capacitance of the gate #3 (CMOS or BiCMOS). Therefore the load capacitance of node (b) is updated and the loop is removed (Fig. 5(B)). Feedback loops are found using a depth first search (DFS) exploration of circuits[6]. The classical DFS uses a visited flag with each node. This flag is set to true each time a node is explored. If the flag is found true before exploration then a feed-back loop is detected. For digital circuits exploration, an additional flag associated with each cell is provided in order to detect multiple output cells within feed-back loops.

4 Algorithm Complexity

The loop elimination and classification phases of the algorithm have both linear running time with the number of cells in a circuit. Therefore each circuit cell is processed only two times: one time by the loop elimination pass and one time by the selection pass.

5 Tests and Discussion

As test vehicles, we have analyzed many different circuits, among them we have chosen several combinational circuits from the ISCAS-85 benchmarks [9]. A 0.8 um mixed CMOS/BiCMOS standard cells library is used for the tests. This library was designed following the strategy defined in Section 2. The load capacitances at the outputs used in these examples, have a value of 0.3 pF. The measured analysis time is the time needed to select all the cells in a block. It is defined as the overall time of the tool minus the time needed to process the netlist and the library. The results obtained on a 25 Mhz R3000 Mips workstation , are shown in Table I.

The achieved speed improvements of the circuits, as compared to their pure CMOS versions range from about 11%, to about 41%. On the other hand, the area penalty for worst conditions (I/O pads area not included), ranges from 6.2% to 35.8%. The area penalty for circuit C17 is the largest one because the number of BiCMOS gates is relatively high (1/3 of the total number of gates). In fact, for such circuits the number of primary outputs is important as compared to the size of the circuit. Since all cells driving primary outputs are BiCMOS ones, the overall number of BiCMOS gates is large and implies an important area overhead. For normal size and large circuits, this overhead is typically of the same order as speed improvement. However, there are circuits where the area penalty is relatively small regarding to speed improvement. One noticeable example is circuit C432. This is due to the fact that for this type of circuits, most of BiCMOS cells are included in the most significant critical paths. This remark suggests that in future versions of our tool, BiCMOS cell replacement should take place exclusively in the critical paths.

Despite the fact that our optimization approach is local, the experience with the tool has shown that generally, the critical paths of the analyzed circuits reach the region of the circuit where our algorithm has performed BiCMOS cell replacement. This is due to the high capacitance of the concerned circuit nodes, which increases the probability that these nodes are included into the critical paths. In this first version of the tool, the cells are selected with an acceptable accuracy, since the computed load capacitance represents the minimum value of the actual load. Thus if the decision of the cell selection is BiCMOS, then Min (C) > Cx. Therefore the exact value of C is greater than Cx. However actual routing capacitances can be ultimately backannotated to the tool in order to lead to accurate final results.

Circuit	Number of circuit gates	Number of BiCMOS gates	Speed Improvement	Area penalty	Analysis Time (ms)
C1355	620	58	11.2%	13.1%	19.1
C17	6	2	14.3%	35.8%	0.5
C1908	442	85	34.6%	15.7%	26.5
C2670	733	135	11.2%	11.4%	22
C432	180	20	40.9%	6.2%	5.7
C499	400	58	19.1%	18.8%	15.1
C880	263	57	12.3%	13.5%	11.9

Table 1

6 Conclusion and Future Work

A CAD tool that automatically selects CMOS/BiCMOS logic gates in a circuit has been developed. It is suitable for intermixing CMOS/BiCMOS cells into a block that contains several hundred cells. Such a tool permits to optimize BiCMOS circuits, and gives a noticeable improvement of speed with minimum overhead of area. The tool is fast, since the average time needed to process one cell is roughly less than 0.1 ms. Moreover, unlike many other optimization tools which assume feed-back loops free logic networks [1,2,8], or which assume that each cell must have only one output [8] , our tool processes any network.

We believe that future extensions of this work should explore user-constrained (timing, area and power constraints) optimization, that consider critical path issue. Also, an extended library wich contains many differently sized versions of each cell, in different technologies such as CMOS and BiCMOS as well as ECL, should be taken into consideration.

References

[1] Shen Lin et al
"Delay and area optimization in standard-cell
design" , 27 th Design Automation Conference, pp 349-352, 1990.

[2] Pak K. Chan
"Algorithms for library specific sizing of
combinational logic" 27 th Design Automation Conference,
pp 353-356, 1990.

[3] Brayton et al
"MIS: A Multiple-level logic optimization
system", IEEE Trans. on CAD, pp 1062-1081, Nov. 1987.

[4] S. H. K. Embabi, A. Bellaouar and M.I.Elmasry
"BiCMOS Digital IC Design", Kluwer Academics Publishers, M.A, 1993.

[5] Hotta et al
"A 70 MHZ 32-b microprocessor with 1.0 um BiCMOS
macrocell library" IEEE Journal of solid-state circuits ,
pp 770-777, June 1990.

[6] Cormen et al
"Introduction to algorithms", MIT Press, 1991.

[7] T. Shinsha et al
"POLARIS: Polarity propagation algorithms
for combinations logic synthesis", 21 th Design Automation
Conference, pp 322-328, 1984.

[8] K.J. Singh et al
"A heuristic algorithm for the fanout problem",
27 th Design Automation Conference, pp 357-360, 1990.

[9] F.Brglez and H.Fujiwara
"A neutral netlist of 10 combinational benchmark circuits and
a target translator in FORTRAN " , ISCAS'85, June 1985.